Computation on Arrays: Broadcasting

- Broadcasting is simply a set of rules for applying binary ufuncs (addition, subtraction, multiplication, etc.) on arrays of different sizes.

Introducing Broadcasting

In [1]:
```
import numpy as np

a = np.array([0,1,2])
b = np.array([5,5,5])
a+b
```

Out[1]:
```
array([5, 6, 7])
```

In [2]:
```
a+np.array([5])
```

Out[2]:
```
array([5, 6, 7])
```

In [3]:
```
a+np.array(5)
```

Out[3]:
```
array([5, 6, 7])
```

In [4]:
```
a+5
```

Out[4]:
```
array([5, 6, 7])
```

Broadcasting

- เมื่อนำอาเรย์ 2 ตัวมาคำนวณแบบ element-wise แต่อาเรย์ทั้งสองมีขนาดไม่เท่ากัน
- ระบบจะ broadcast อาเรย์ตัวเล็ก (หรืออาจทำทั้งสองตัว) ให้มีขนาดเท่ากันก่อนทำงาน

$$\begin{bmatrix}2 & 3\end{bmatrix} + \begin{bmatrix}1 & 2\\ 3 & 4\\ 5 & 6\end{bmatrix} \Rightarrow \begin{bmatrix}2 & 3\\ 2 & 3\\ 2 & 3\end{bmatrix} + \begin{bmatrix}1 & 2\\ 3 & 4\\ 5 & 6\end{bmatrix}$$

- แต่บางครั้งก็ broadcast ไม่ได้

$$\begin{bmatrix}2 & 3 & 4\end{bmatrix} + \begin{bmatrix}1 & 2\\ 3 & 4\\ 5 & 6\end{bmatrix} \qquad \begin{bmatrix}2\\ 3\\ 4\end{bmatrix} + \begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{bmatrix}$$

ตัวอย่าง: broadcast ตัวเล็ก ให้เท่าตัวใหญ่

```
x = np.array([[1,2],[3,4],[5,6]])
u = np.array([2]) + x
```

$$\begin{bmatrix}2\end{bmatrix} + \begin{bmatrix}1 & 2\\ 3 & 4\\ 5 & 6\end{bmatrix}$$

```
(1)
(3, 2)
```

Loading [MathJax]/extensions/Safe.js

ตัวอย่าง: broadcast ตัวเล็ก ให้เท่าตัวใหญ่

```
x = np.array([[1,2],[3,4],[5,6]])
u = np.array([2]) + x
```

$$\begin{bmatrix} 2 & 2 \\ 2 & 2 \\ 2 & 2 \end{bmatrix} + \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

ตัวอย่าง: broadcast ตัวเล็ก ให้เท่าตัวใหญ่

```
x = np.array([[1,2],[3,4],[5,6]])
u = np.array([2]) + x
```

$$\begin{bmatrix} 2+1 & 2+2 \\ 2+3 & 2+4 \\ 2+5 & 2+6 \end{bmatrix}$$

ตัวอย่าง: broadcast ตัวเล็ก ให้เท่าตัวใหญ่

```
x = np.array([[1,2],[3,4],[5,6]])
w = np.array([10 20]) + x
```

$$[10 \quad 20] + \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

```
   (2)
(3, 2)
```

ตัวอย่าง: broadcast ตัวเล็ก ให้เท่าตัวใหญ่

```
x = np.array([[1,2],[3,4],[5,6]])
w = np.array([10 20]) + x
```

$$\begin{bmatrix} 10 & 20 \\ 10 & 20 \\ 10 & 20 \end{bmatrix} + \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

Loading [MathJax]/extensions/Safe.js

ตัวอย่าง: broadcast ตัวเล็ก ให้เท่าตัวใหญ่

```
x = np.array([[1,2],[3,4],[5,6]])
w = np.array([10 20]) + x
```

$$\begin{bmatrix} 10+1 & 20+2 \\ 10+3 & 20+4 \\ 10+5 & 20+6 \end{bmatrix}$$

ตัวอย่าง: broadcast ตัวเล็ก ให้เท่าตัวใหญ่

```
x = np.array([[1,2],[3,4],[5,6]])
v = np.array([[10],[20],[30]]) + x
```

$$\begin{bmatrix} 10 \\ 20 \\ 30 \end{bmatrix} + \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

```
(3, 1)
(3, 2)
```

Loading [MathJax]/extensions/Safe.js

ตัวอย่าง: broadcast ตัวเล็ก ให้เท่าตัวใหญ่

```
x = np.array([[1,2],[3,4],[5,6]])
v = np.array([[10],[20],[30]]) + x
```

$$\begin{bmatrix} 10 & 10 \\ 20 & 20 \\ 30 & 30 \end{bmatrix} + \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

ตัวอย่าง: broadcast ตัวเล็ก ให้เท่าตัวใหญ่

```
x = np.array([[1,2],[3,4],[5,6]])
v = np.array([[10],[20],[30]]) + x
```

$$\begin{bmatrix} 10+1 & 10+2 \\ 20+3 & 20+4 \\ 30+5 & 30+6 \end{bmatrix}$$

ตัวอย่าง: broadcast ทั้ง 2 ตัว

$$\begin{bmatrix}1\\2\\3\end{bmatrix} + \begin{bmatrix}4 & 5\end{bmatrix}$$

```
(3, 1)
   (2)
```

ตัวอย่าง: broadcast ทั้ง 2 ตัว

$$\begin{bmatrix}1\\2\\3\end{bmatrix} + \begin{bmatrix}4 & 5\end{bmatrix} \Rightarrow \begin{bmatrix}1 & 1\\2 & 2\\3 & 3\end{bmatrix} + \begin{bmatrix}4 & 5\end{bmatrix}$$

```
(3, 1)          (3, 2)
   (2)             (2)
```

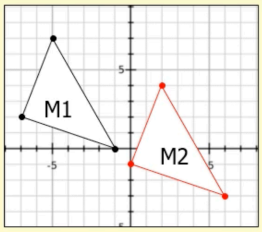
ตัวอย่าง: broadcast ทั้ง 2 ตัว

$$\begin{bmatrix}1\\2\\3\end{bmatrix} + \begin{bmatrix}4 & 5\end{bmatrix} \Rightarrow \begin{bmatrix}1 & 1\\2 & 2\\3 & 3\end{bmatrix} + \begin{bmatrix}4 & 5\end{bmatrix} \Rightarrow \begin{bmatrix}1 & 1\\2 & 2\\3 & 3\end{bmatrix} + \begin{bmatrix}4 & 5\\4 & 5\\4 & 5\end{bmatrix}$$

```
(3, 1)      (3, 2)      (3, 2)
  (2)         (2)       (3, 2)
```

ตัวอย่าง: Matrix Translation

```
   x  y
[[-7  2]
 [-5  7]
 [-1  0]]
```




```
[[ 0 -1]
 [ 2  4]
 [ 6 -3]]
```

ต้องการย้ายทุกจุดไปทางขวา 7, ลงล่าง 3 คือบวกทุดจุดด้วย [7, -3]

M1 กับ T มีขนาดเท่ากัน

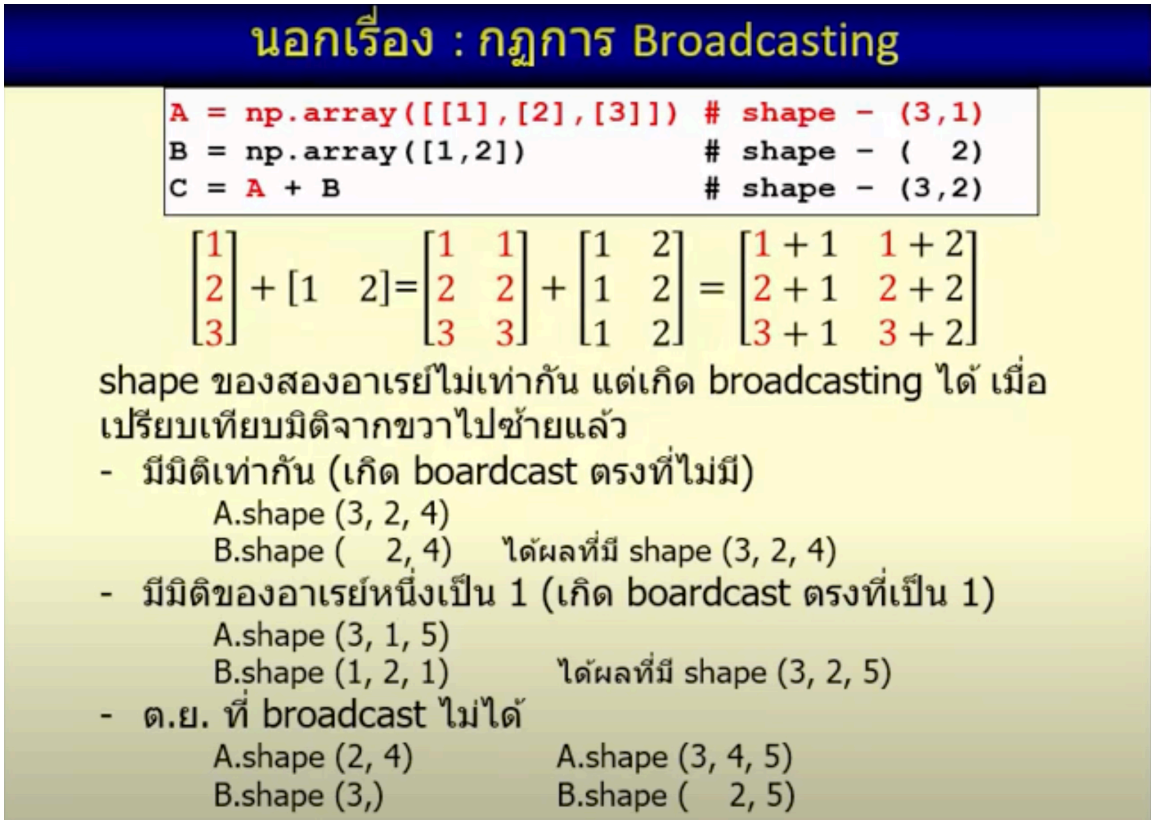
```
M1 = np.array([ [-7,2],[-5,7],[-1,0] ])
T = np.array([ [7,-3],[7,-3],[7,-3] ])
M2 = M1 + T
```

เขียนแค่นี้ก็พอ

```
M2 = M1 + np.array( [7, -3] )
```

```
[[-7  2]                [[-7  2]     [[7  -3]
 [-5  7]  + [7, -3]  ⇒   [-5  7]  +   [7  -3]
 [-1  0]]                [-1  0]]     [7  -3]]
```

Loading [MathJax]/extensions/Safe.js



In [5]:
```
# We can similarly extend this to arrays of higher dimension. Observe the re
# we add a one-dimensional array to a two-dimensional array:

M = np.ones((3,3))
M
```

Out[5]:
```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

In [6]:
```
a = np.array([0,1,2])
a
```

Out[6]:
```
array([0, 1, 2])
```

In [7]:
```
M+a
# Here the one-dimensional array a is stretched, or broadcast, across the se
# dimension in order to match the shape of M .
```

Out[7]:
```
array([[1., 2., 3.],
       [1., 2., 3.],
       [1., 2., 3.]])
```

In [8]:
```
c = np.arange(3).reshape((3,1))
c
```

Out[8]:
```
array([[0],
       [1],
       [2]])
```

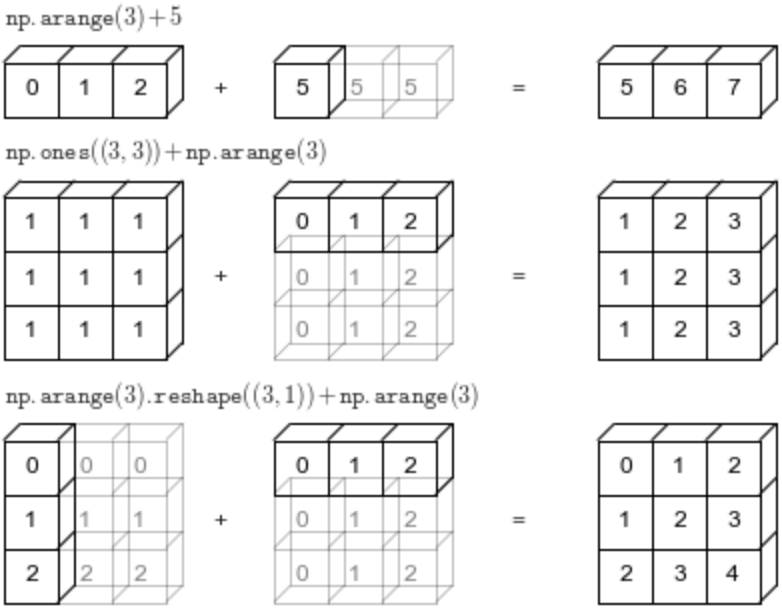
In [9]:
```
d = np.arange(3)
d
```

Out[9]:
```
array([0, 1, 2])
```

In [10]:
```
c+d
```

Out[10]:
```
array([[0, 1, 2],
       [1, 2, 3],
       [2, 3, 4]])
```

Visualization of NumPy broadcasting



Comparisons, Masks, and Boolean Logic

Comparison Operators as ufuncs

- The result of these comparison operators is always an array with a Boolean data type. All six of the standard comparison operations are available:
- for example, you might wish to count all values greater than a certain value, or perhaps remove all outliers that are above some threshold. In NumPy, Boolean masking is often the most efficient way to accomplish these types of tasks.

In [11]:
```
x = np.array([1,2,3,4,5])

print(x<3)  # less than
print(x>3)  # greater than
print(x<=3) # less than or equal
print(x>=3) # greater than or equal
print(x!=3) # not equal
print(x==3) # equal
```

```
[ True  True False False False]
[False False False  True  True]
[ True  True  True False False]
[False False  True  True  True]
[ True  True False  True  True]
[False False  True False False]
```

In [12]:
```
# It is also possible to do an element-by-element comparison of two arrays,
# include compound expressions:

(2*x) == (2**x)
```

Out[12]:
```
array([ True,  True, False, False, False])
```

In [13]:
```
# As in the case of arithmetic operators, the comparison operators are imple
# ufuncs in NumPy; for example, when you write x < 3 , internally NumPy uses
# np.less(x, 3) . A summary of the comparison operators and their equivalent
# is shown here:
```

Comparison operators and their equivalent

Operator	Equivalent ufunc
==	np.equal
!=	np.not_equal
<	np.less
<=	np.less_equal
>	np.greater
>=	np.greater_equal

In [14]:
```
x = np.random.randint(10, size=(3,4))
print(x)

x<6
```

```
[[2 6 5 0]
 [7 9 4 0]
```

Out[14]:
```
array([[ True, False,  True,  True],
       [False, False,  True,  True],
       [ True, False,  True,  True]])
```

Working with Boolean Arrays

In [15]:
```
print(x)

# To count the number of True entries in a Boolean array,
#     np.count_nonzero is useful:
# np.count_nonzero เป็นฟังก์ชันใน NumPy ที่ใช้สำหรับนับจำนวนของสมาชิกใน array
#     ที่มีค่า " ไม่เท่ากับ 0" หรือในทางอื่นคือ นับจำนวนสมาชิกที่เป็น "True"
#     ในบริบทของการตีความเป็น Boolean(ซึ่ง 0 ถูกมองว่าเป็น False
#     และค่าที่ไม่ใช่ 0 ถูกมองว่าเป็น True)

# how many values less than 6?
print("A: ",np.count_nonzero(x<6))

# We see that there are eight array entries that are less than 6.
#     Another way to get at this information is to use np.sum ;
#     in this case, False is interpreted as 0 ,
#     and True is interpreted as 1 :

print("B: ",np.sum(x<6))

print("C: ",np.sum(x!=np.nan))
print("D: ",np.count_nonzero(x!=np.nan))
```

```
[[2 6 5 0]
 [7 9 4 0]
 [4 6 4 2]]
A:  8
B:  8
C:  12
D:  12
```

In [16]:
```
# how many values less than 6 in each row?
print(np.sum(x < 6, axis=1))

# how many values less than 6 in each column?
print(np.sum(x < 6, axis=0))
```

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

In [17]:
```
# If we're interested in quickly checking whether any or all
#     the values are true, we can use (you guessed it)
#     np.any() or np.all() :

# are there any values greater than 8?
print(np.any(x>8))

# e any values less than zero?
print(np.any(x<0))
```

```
# are all values less than 10?
print(np.all(x<10))

# are all values equal to 6?
print(np.all(x==6))
```

```
True
False
True
False
```

In [18]:
```
# are all values in each row less than 8?
print(np.all(x<8, axis=1))

# are all values in each column less than 3?
print(np.all(x<3, axis=0))
```

```
[ True False  True]
[False False False  True]
```

In [19]:
```
print(x)
print(x<5)
print(x[x<5])
```

```
[[2 6 5 0]
 [7 9 4 0]
 [4 6 4 2]]
[[ True False False  True]
 [False False  True  True]
 [ True False  True  True]]
[2 0 4 0 4 4 2]
```

In [20]:
```
# In Python, all nonzero integers will evaluate as True .
bool(42), bool(0), bool(-1)
```

Out[20]: (True, False, True)

In [21]:
```
bool(42 and 0)
```

Out[21]: False

In [22]:
```
bool(42 or 0)
```

Out[22]: True

In [23]:
```
# When you have an array of Boolean values in NumPy, this can be thought of
# string of bits where 1 = True and 0 = False , and the result of & and | op
# similar manner as before:

A = np.array([1, 0, 1, 0, 1, 0], dtype=bool)
B = np.array([1, 1, 1, 0, 1, 1], dtype=bool)
A | B
```

Out[23]: array([True, True, True, False, True, True])

In [24]:
```
x = np.arange(10)
(x > 4) & (x < 8)
```

Out[24]:
```
array([False, False, False, False, False,  True,  True,  True, False,
       False])
```

Fancy Indexing

Exploring Fancy Indexing

การเลือกสมาชิกด้วย array ของดัชนีใน array 1 มิติ

In [25]:
```
import numpy as np

# สร้าง array 1 มิติ
a = np.random.randint(100, size=10)

# ใช้ fancy indexing โดยส่ง array ของดัชนีที่ต้องการดึงข้อมูล
indices = [1, 8, 5, 3] # หรืออาจใช้ ind = [1, 8, 5, 3]
result = a[indices] # หรืออาจใช้ ind

print("Array ต้นฉบับ:", a)
print("Indices ที่เลือก:", indices)
print("ผลลัพธ์จาก Fancy Indexing:", result)
```

```
Array ต้นฉบับ: [67  8 61 27 49 68  6 79 55 59]
Indices ที่เลือก: [1, 8, 5, 3]
ผลลัพธ์จาก Fancy Indexing: [ 8 55 68 27]
```

In [26]:
```
indices = np.array([[3, 7],
                    [4, 5]])
a[indices]
```

Out[26]:
```
array([[27, 79],
       [49, 68]])
```

Fancy Indexing กับ array แบบหลายมิติ

In [27]:
```
import numpy as np

# สร้าง array 2 มิติ (matrix)
X = np.array([[ 1,  2,  3,  4],
              [ 5,  6,  7,  8],
              [ 9, 10, 11, 12],
              [13, 14, 15, 16]])

# ถ้าต้องการเลือกสมาชิกในตำแหน่งที่เฉพาะเจาะจง
# เช่น เลือกสมาชิกที่อยู่ในตำแหน่ง (0,1), (1,2), (2,3), (3,0)
```

```
row_ind = np.array([0, 1, 2, 3])
col_ind = np.array([1, 2, 3, 0])

result = X[row_ind, col_ind]

print("X:")
print(X)
print("Row ind:", row_ind)
print("Column ind:", col_ind)
print("ผลลัพธ์ Fancy Indexing (เลือกสมาชิกตามตำแหน่ง):", result)
```

```
X:
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]]
Row ind: [0 1 2 3]
Column ind: [1 2 3 0]
ผลลัพธ์ Fancy Indexing (เลือกสมาชิกตามตำแหน่ง): [ 2  7 12 13]
```

In [28]:
```
result2 = X[row_ind.reshape((4,1)), col_ind]

print("X:")
print(X)
print("Row indices2:")
print(row_ind.reshape((4,1)))
print("Column indices:", col_ind)
print("ผลลัพธ์ Fancy Indexing (เลือกสมาชิกตามตำแหน่ง):")
print(result2)
```

```
X:
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]]
Row indices2:
[[0]
 [1]
 [2]
 [3]]
Column indices: [1 2 3 0]
ผลลัพธ์ Fancy Indexing (เลือกสมาชิกตามตำแหน่ง):
[[ 2  3  4  1]
 [ 6  7  8  5]
 [10 11 12  9]
 [14 15 16 13]]
```

Combined Indexing

In [29]:
```
print(X)
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]]
```

In [30]:
```
X[2,[2,0,1]]
```

Out[30]:
```
array([11,  9, 10])
```

In [31]:
```
X[1:, [2, 0, 1]]
```

Out[31]:
```
array([[ 7,  5,  6],
       [11,  9, 10],
       [15, 13, 14]])
```

Modifying Values with Fancy Indexing

In [32]:
```
x = np.arange(10)
i = np.array([2,1,8,4])
x[i] = 99
print(x)
```

```
[ 0 99 99  3 99  5  6  7 99  9]
```

In [33]:
```
x[i] -= 10
print(x)
```

```
[ 0 89 89  3 89  5  6  7 89  9]
```

In [34]:
```
x = np.zeros(10)
x[[0, 2]] = [4, 6]
print(x)
```

```
[4. 0. 6. 0. 0. 0. 0. 0. 0. 0.]
```

In [35]:
```
x = np.zeros(10)
x[[0, 0]] = [4, 6]
print(x)

# Where did the 4 go?
# The result of this operation is to first assign x[0] = 4,
#     followed by x[0] = 6 .
# The result, of course, is that x[0] contains the value 6.
```

```
[6. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

การใช้ Boolean array สำหรับการเลือกข้อมูล (อีกรูปแบบของ Fancy Indexing)

In [36]:
```
import numpy as np

# สร้าง array 1 มิติ
a = np.random.randint(100, size=10)

# ...lean array โดยใช้เงื่อนไข เช่น เลือกค่าที่มากกว่า 40
```

```
bool_index = a > 40
result = a[bool_index]

print("Array ต้นฉบับ:", a)
print("Boolean Index (a > 40):")
print(bool_index)
print("ผลลัพธ์จาก Fancy Indexing โดยใช้ Boolean array:")
print(result)
```

```
Array ต้นฉบับ: [73 14 79 57 75 28 57 27 37 73]
Boolean Index (a > 40):
[ True False  True  True  True False  True False False  True]
ผลลัพธ์จาก Fancy Indexing โดยใช้ Boolean array:
[73 79 57 75 57 73]
```

ในตัวอย่างนี้ เราสร้าง Boolean array ที่มีค่า True สำหรับตำแหน่งที่สมาชิกใน a มีค่ามากกว่า 40 แล้วใช้ Boolean array นี้ในการเข้าถึงสมาชิกใน a ผลลัพธ์จะได้ array ที่ประกอบด้วยสมาชิกที่ผ่านเงื่อนไข

Sorting Arrays

Fast Sorting in NumPy: np.sort and np.argsort

In [37]:
```
x = np.array([2,1,4,3,5])
np.sort(x)
```

Out[37]:
```
array([1, 2, 3, 4, 5])
```

In [38]:
```
x.sort()
print(x)
```

```
[1 2 3 4 5]
```

In [39]:
```
#return indices
x = np.array([2,1,4,3,5])
i = np.argsort(x)
print(i)

x[i]
```

```
[1 0 3 2 4]
```

Out[39]:
```
array([1, 2, 3, 4, 5])
```

In [40]:
```
y = np.random.randint(100, size=10)
y
```

```
array([74, 97, 80,  2, 15,  2, 24, 69, 97, 82])
```

In [41]:
```
np.sort(y)
```

Out[41]:
```
array([ 2,  2, 15, 24, 69, 74, 80, 82, 97, 97])
```

In [42]:
```
y.sort()
print(y)
```

```
[ 2  2 15 24 69 74 80 82 97 97]
```

Sorting along rows or columns

In [43]:
```
# A useful feature of NumPy's sorting algorithms is the
#     ability to sort along specific rows or columns of
#     a multidimensional array using the axis argument.
# For example:

X = np.random.randint(0,10,(4,6))
print(X)
```

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

In [44]:
```
# sort each column of X

np.sort(X, axis=0)
```

Out[44]:
```
array([[1, 0, 7, 0, 0, 4],
       [4, 5, 7, 3, 4, 4],
       [4, 6, 8, 8, 5, 5],
       [9, 8, 9, 9, 8, 8]])
```

In [45]:
```
# sort each row of X

np.sort(X, axis=1)
```

Out[45]:
```
array([[4, 4, 5, 7, 8, 8],
       [0, 3, 5, 7, 8, 9],
       [0, 0, 1, 4, 6, 9],
       [4, 4, 5, 8, 8, 9]])
```

Partial Sorts: Partitioning

In [46]:
```
# Note that the first four values in the resulting array
#    are the four smallest in the array, and the remaining
#    array positions contain the remaining values.
#    Within the two partitions, the elements have arbitrary order.
```

```
In [46]:
x = np.array([7, 2, 1, 3, 6, 5, 4])
np.partition(x, 4)
```

```
Out[46]: array([2, 3, 1, 4, 5, 6, 7])
```

```
In [47]:
y = np.random.randint(100, size=10)
y
```

```
Out[47]: array([51, 87, 60, 14, 96, 59, 99, 41, 77, 22])
```

```
In [48]:
np.partition(y, 4)
```

```
Out[48]: array([14, 22, 41, 51, 59, 60, 77, 87, 96, 99])
```

```
In [49]:
print(X)
```

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

```
In [50]:
# The result is an array where the first two slots in each row contain the s
# from that row, with the remaining values filling the remaining slots.

np.partition(X, 2, axis=1)
```

```
Out[50]: array([[4, 4, 5, 8, 7, 8],
                [0, 3, 5, 9, 8, 7],
                [0, 0, 1, 9, 6, 4],
                [4, 4, 5, 9, 8, 8]])
```

```
In [51]:
np.partition(X, 2, axis=0)
```

```
Out[51]: array([[1, 0, 7, 0, 0, 4],
                [4, 5, 7, 3, 4, 4],
                [4, 6, 8, 8, 5, 5],
                [9, 8, 9, 9, 8, 8]])
```

```
In [52]:
np.argpartition(X, 2, axis=1)
```

```
Out[52]: array([[0, 5, 4, 3, 2, 1],
                [1, 3, 5, 0, 4, 2],
                [3, 4, 0, 2, 1, 5],
                [0, 4, 1, 3, 2, 5]])
```

```
In [53]:
np.argpartition(X, 2, axis=0)
```

```
Out[53]: array([[2, 1, 0, 2, 2, 0],
                [0, 3, 1, 1, 3, 2],
                [3, 2, 3, 0, 0, 1],
                [1, 0, 2, 3, 1, 3]])
```

```
In [ ]:
```