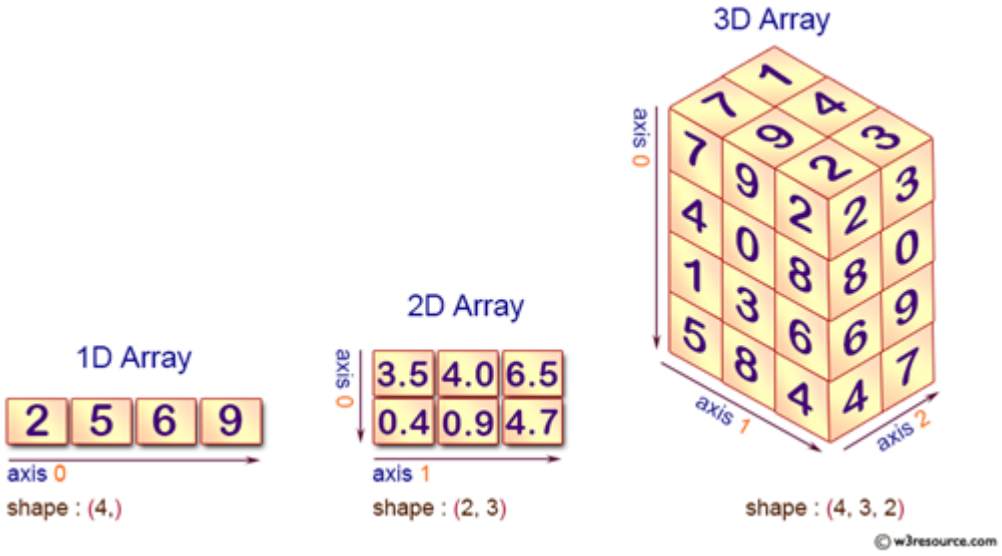
ndarray

***Array** o **vettore** o **1D array**: è un insieme di dati tutti dello stesso tipo. Il numero di elementi è detto **dimensione** o **lunghezza** del vettore. Ogni elemento è individuabile tramite un **indice**, comunemente questo indice parte da 0. Es. a=[1,2,3] ha dimensione 3, l'elemento di indice 1 si indica con a[1] ed è uguale a 2.*

***Matrice: vettore bidimensionale o a due dimensioni** o **2D array**. Gli elementi sono organizzati in righe e colonne. Es. a=[[1,2], [10,20], [100,200]] ha 2 dimensioni: 3 righe e 2 colonne. L'elemento a[2,1] vale 200.*

***Array multidimensionale** o **N-dimensional array**: array a N dimensioni, con N>2 si parla di **tensori**. Es. a = [[[1,2], [10,20], [100,200]], [[3,4], [30,40], [300,400]]] ha 3 dimensioni che valgono 2,3,2; l'elemento a[0,1,0] contiene 10.*

***Asse** o **axis**: in Numpy, ogni dimensione corrisponde ad un asse. Nella matrice dell'esempio l'asse 0 corrisponde alle righe e ha lunghezza 3, mentre l'asse 1 corrisponde alle colonne e ha lunghezza 2.*



Tipi in Numpy

	8 bit	16 bit	32 bit	64 bit	128 bit
int	int8	int16	int32	int64	
uint	uint8	uint16	uint32	uint64	
float		float16	float32	float64	float128

Creare ndarray

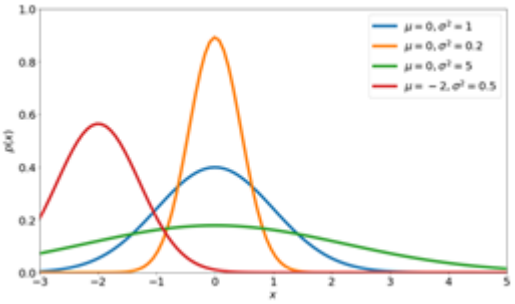
```
import numpy as np

a1 = np.array((1,2,3))              #array da tupla
a2 = np.array([1.6,-1,3], np.uint8)#array da lista con specifica di tipo
a3 = np.arange(10, dtype = float)  #array da range [0..stop) con passo 1 e specifica di tipo
a4 = np.arange(4,10,2)             #array da range [start..stop) con passo 2
a5 = np.linspace(0,10,5)           #array di 5 numeri equamente distribuiti in [start..stop],
                                   #si può usare dtype
print("a1:",a1); print("a2:",a2); print("a3:",a3); print("a4:",a4); print("a5:",a5)
```

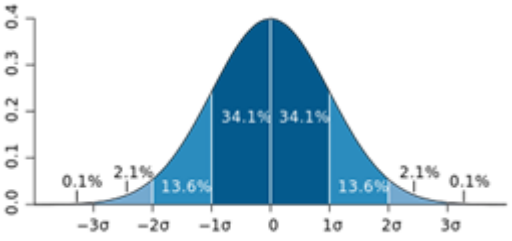
```
a1: [1 2 3]
a2: [  1 255   3]
a3: [0. 1. 2. 3. 4. 5. 6. 7. 8. 9.]
a4: [4 6 8]
a5: [ 0.   2.5  5.   7.5 10. ]
```

```
a6 = np.zeros(5, dtype = np.uint) # array di 0 con specifica di tipo (per default float64)
a7 = np.ones(5)                   # array di 1 (per default float64)
a8 = np.zeros((2,3))              # array di 0 con dimensioni specificate
print(a6); print(a7); print(a8);
a8.fill(3)                        # l'array deve esistere già
print(a8)
```

```
[0 0 0 0 0]
[1. 1. 1. 1. 1.]
[[0. 0. 0.]
 [0. 0. 0.]]
[[3. 3. 3.]
 [3. 3. 3.]]
```

- ***Distribuzione Gaussiana o normale**: molti fenomeni naturali hanno una distribuzione dei valori che segue una forma a campana, per cui la maggioranza dei valori è concentrata intorno ad un **valore medio**μ (mu).*
- *La **deviazione standard** σ (sigma) esprime la dispersione dei valori attorno a questo valore medio, un valore piccolo indica che i campioni si discostano poco dal valore medio e quindi la campana e molto ripida, valori maggiori indicano valori più distribuiti e quindi una campana più "larga". In ogni caso il 68,26% dei campioni si troverà tra $\mu+\sigma$ e $\mu-\sigma$ e ben il 99,75% (praticamente tutti) tra $\mu+3\sigma$ e $\mu-3\sigma$.*
- *La **varianza** σ^2 equivale al quadrato della deviazione standard e quindi esprime di quanto, in media, i campioni si discostano (quadraticamente) dalla media aritmetica, ovvero*

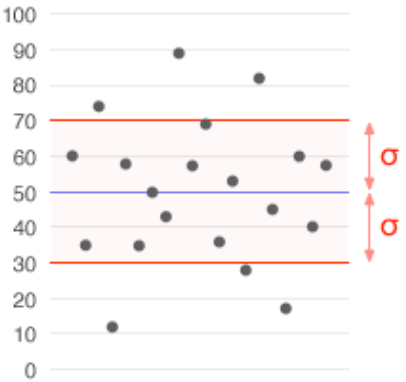




*ESEMPIO: Se mediamente in Svezia si leggono 50 libri con una deviazione standard di 20, si ha la **certezza matematica** che circa il 68% degli svedesi **leggono tra i 30 e i 70 libri** l'anno*

media - deviazione standard

media + deviazione standard



*e che il 95% **tra i 10 e 90 libri***

*media - deviazione standard * 2*

*media + deviazione standard *2*

Creiamo array con valori casuali

```
mat1 = np.random.rand(2,3)               # matrice 2x3 di valori casuali reali in [0..1)
mat2 = np.random.randint(1,7, size=(2,3))# matrice 2x3 di valori casuali interi in [1..7)
mat3 = np.random.randn(2,3)              # matrice 2x3 di valori casuali reali campionati
                                         # da una distribuzione normale con m = 0 e dev = 1
print("rand(2,3):\n", mat1);
print("randint(1,7, size=(2,3)):\n", mat2);
print("randn(2,3):\n", mat3);
```

```
rand(2,3):
 [[0.17257134 0.79468453 0.31987684]
 [0.06160697 0.7873164  0.15297448]]
randint(1,7, size=(2,3)):
 [[6 3 4]
 [3 5 1]]
randn(2,3):
 [[ 2.19879202 -0.36679545  0.64247894]
 [ 1.46445172  1.68563218 -0.31445489]]
```

per generare dei numeri casuali secondo una distribuzione gaussiana con m=-2 e dev= 0.5

```
matNorm = np.random.normal(-2, 0.5, (2,3))
print(matNorm)
```

```
[[-1.26426307 -1.66506118 -1.59426107]
 [-1.92511942 -2.16551803 -2.43796502]]
```

Per ottenere dei numeri reali pseudo casuali tra un valore min (incluso) e un valore max(escluso) possiamo usare la formula

```
numpy.random.rand() * (max - min + 1) + min
```

oppure la funzione che genera i numeri secondo una distribuzione uniforme (ogni numero ha la stessa probabilità di uscire)

```
numpy.random.uniform(min,  max)
```

Lancio con moneta truccata

```
print(np.random.choice(['testa', 'croce'], size=8, p=[0.8, 0.2]))
```

```
['testa' 'testa' 'croce' 'croce' 'testa' 'testa' 'testa' 'testa']
```

Per decidere i turni (`replace=False` per non rimettere ii valori nel "sachetto" da cui si estrae)

```
print(np.random.choice(['Qui', 'Quo', 'Qua'], size=3, replace=False))
```

```
['Qui' 'Qua' 'Quo']
```

Ogni volta che si esegue una delle funzioni sopraindicate, otteniamo un diverso set di numeri casuali. Quando addestreremo un modello di machine learning, avremo bisogno di generare lo stesso insieme di numeri casuali ogni volta per poter fare dei confronti. Per fare questo imposteremo il seme del generatore pseudocasuale ad un valore fisso scrivendo:

```
np.random.seed(100)   #il valore può essere scelto a piacere
```

Se aggiungiamo questa istruzione all'inizio di una cella e la eseguiamo più volte, l'output ottenuto sarà sempre lo stesso.

La forma degli ndarray

```
ndarray = np.array([[[1,2], [10,20], [100,200]]])
print(ndarray)
print("numero di dimensioni:", np.ndim(ndarray))          # anche ndarray.ndim
print("elementi per ogni dimensione:", np.shape(ndarray)) # anche ndarray.shape
print("elementi totali:", np.size(ndarray))               # anche ndarray.size
print("tipo elementi:", ndarray.dtype)
```

```
[[[  1   2]
  [ 10  20]
  [100 200]]]
numero di dimensioni: 3
elementi per ogni dimensione: (1, 3, 2)
elementi totali: 6
tipo elementi: int64
```

```
array1 = ndarray.flatten()       # crea una copia a 1 dimensione del corrispondente array
array2 = ndarray.ravel()         # mantiene un riferimento con l'array originale
array1[0] = 0
array2[1] = 0
print("array1: \n", array1); print("array1: \n", array2);
print("origine: \n",ndarray)   # modificato solo "attraverso" ravel
```

```
array1:
 [  0   2  10  20 100 200]
array1:
 [  1   0  10  20 100 200]
origine:
 [[[  1   0]
  [ 10  20]
  [100 200]]]
```

```
array1.shape=(3,2)              # modifica la struttura
print(array1)
mat = ndarray.reshape(2,-1)     # restituisce un ndarray con struttura modificata
print(mat)
```

```
[[  0   2]
 [ 10  20]
 [100 200]]
[[  1   0  10]
 [ 20 100 200]]
```

```
m = np.arange(24)           # vettore con elementi da 0 a 23
m = m.reshape(4,6)
print("m.shape",m.shape,"\n", m)
print("transpose.shape",np.transpose(m).shape)# inverte lo shape; per le matrici fa la trasposta
print(m.T)
```

```
m.shape (4, 6)
 [[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]
 [12 13 14 15 16 17]
 [18 19 20 21 22 23]]
transpose.shape (6, 4)
[[ 0  6 12 18]
 [ 1  7 13 19]
 [ 2  8 14 20]
 [ 3  9 15 21]
 [ 4 10 16 22]
 [ 5 11 17 23]]
```

Per fare uno spostamento degli elementi come in uno shift

```
print("axis=None \n",np.roll(m,2))          # viene prima fatto il flatten e poi lo shift
print("axis=0 \n",np.roll(m,1, axis=0))     # shift delle intere righe
print("axis=1 \n",np.roll(m,-1, axis=1))    # shift a sinistra in ciascuna riga
```

```
axis=None
 [[22 23  0  1  2  3]
 [ 4  5  6  7  8  9]
 [10 11 12 13 14 15]
 [16 17 18 19 20 21]]
axis=0
 [[18 19 20 21 22 23]
 [ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]
 [12 13 14 15 16 17]]
axis=1
 [[ 1  2  3  4  5  0]
 [ 7  8  9 10 11  6]
 [13 14 15 16 17 12]
 [19 20 21 22 23 18]]
```

Indicizzazione

Per accedere agli elementi possiamo utilizzare le [ ], ma con alcune differenza rispetto alle liste di Python.

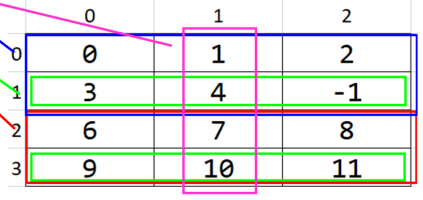
```
a = np.arange(12)    # vettore con elementi da 0 a 11
a = a.reshape(4,-1)  # lo trasforma in una matrice 4x3
print("a:\n",a)
ls = a.tolist()      # restituisce la lista Python corrispondente all'array
print("a.tolist():",ls)

a[1,2] = -1
ls[1][2] = -2        # anche in Numpy si può usare questa notazione, ma è meno efficiente
print("a[1]:", a[1])  # seleziona l'elemento di indice 1, ovvero l'intera riga
print("ls[1]:", ls[1])# seleziona l'elemento di indice 1, ovvero l'intera riga
```

```
a:
 [[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
a.tolist(): [[0, 1, 2], [3, 4, 5], [6, 7, 8], [9, 10, 11]]
a[1]: [ 3  4 -1]
ls[1]: [3, 4, -2]
```

SLICING (crea un riferimento)

```
print(a[:-2])   # da 0 al penultimo elemento escluso
print(a[2:4])   # dall'elemento di indice 2 al 4 escluso
print(a[1:4:2]) # dall'elemento di indice 1 al 4 escluso, passo 2
print(a[:,1:2]) # tutte le righe, dalla colonna 1 alla 2 esclusa
print(a[:,1])   # tutti gli elementi della colonna di indice 1
```

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



Vettori di indici o fancy indexing

```
print(a[[1,2,3]])               # seleziona gli elementi con quell'indice: sono righe
print(a.flatten()[[1,2,3]])     # seleziona gli elementi con quell'indice: ora sono valori
print(a[[1,2,3],0])             # seleziona gli elementi [1,0] [2,0] [3,0]
print(a[[1,2,3],[0,1,2]])       # seleziona gli elementi [1,0] [2,1] [3,2]
```

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

Vettori di booleani

```
print("a[[True, False, True, False]]")
print(a[[True, False, True, False]])            # seleziona solo gli elementi (righe) con True
filtro = np.random.randint(0,2,(4,3))==True     # matrice 4x3 di valori casuali True/False
print("filtro \n",filtro)
print("a[filtro]", a[filtro])
print("a > 5 \n", a > 5)                        # restituisce un array di boolean con la stessa forma di a
print("a[a > 5]", a[a > 5])                     # seleziona gli elementi che soddisfano la condizione
print("logical_not",a[np.logical_not(a <= 5)])  # seleziona gli elementi che NON soddisfano la condizione
print("isin",a[np.isin(a,[7,4,1,8])])           # seleziona gli elementi che sono nell'elenco
```

```
a[[True, False, True, False]]
[[0 1 2]
 [6 7 8]]
filtro
 [[ True  True False]
 [ True  True  True]
 [False  True False]
 [False  True False]]
a[filtro] [ 0  1  3  4 -1  7 10]
a > 5
 [[False False False]
 [False False False]
 [ True  True  True]
 [ True  True  True]]
a[a > 5] [ 6  7  8  9 10 11]
logical_not [ 6  7  8  9 10 11]
isin [1 4 7 8]
```

Riferimento o copia?

```
ar = np.arange(10)
print(ar)
slice_index = ar[0:3]                          # [0 1 2]
fancy_index = ar[[0,1,2]]                      # [0 1 2]
bool_index = ar[np.logical_and(ar>=0,ar<3)]    # [0 1 2]
slice_index[0] = -1                            # modificherà anche ar
fancy_index[1] = -1                            # non modificherà ar
bool_index[2] = -1                             # non modificherà ar
print(ar)
```

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

argwere()

```
ar = np.arange(6)
print(ar)
print(np.argwhere(ar % 2 == 0))    # indici degli elementi pari su 1D array
print("mat")
mat = ar.reshape(2,3)
print(mat)
print(np.argwhere(mat % 2 == 0))   # indice degli elementi pari su 2D array
```

```
[0 1 2 3 4 5]
[[0]
 [2]
 [4]]
mat
[[0 1 2]
 [3 4 5]]
[[0 0]
 [0 2]
 [1 1]]
```

Operazioni con ndarray

```
list = [1,2,3]
arr = np.array(list)

print("list +", list + list) # l'operatore + coincide con la concatenazione
print("array +", arr + arr)  # l'operatore + è applicato elemento per elemento: ufunc
print("list *", list * 2)    # raddoppia la lista
print("array *", arr * 2)    # raddoppia ogni elemento dell'array
```

```
list + [1, 2, 3, 1, 2, 3]
array + [2 4 6]
list * [1, 2, 3, 1, 2, 3]
array * [2 4 6]
```

per concatenare

```
a1 = np.array([[0, 1], [2, 3]])
a2 = np.array([[4, 5]])

print("axis=0 \n", str(np.concatenate((a1, a2), axis=0)))             # di default
print("axis=1 \n", str(np.concatenate((a1, a2.transpose()), axis=1)))
print("axis=None\n", str(np.concatenate((a1, a2), axis=None)))        # flatten
```

```
axis=0
 [[0 1]
 [2 3]
 [4 5]]
axis=1
 [[0 1 4]
 [2 3 5]]
axis=None
 [0 1 2 3 4 5]
```

```
a1 = np.array([0, 1])
a2 = np.array([2, 3])

print("axis=0 \n", np.stack((a1, a2)))             # sottinteso axis=0
print("axis=1 \n", np.stack((a1, a2), axis=-1))    # ultimo asse ovvero axis=1
```

```
axis=0
 [[0 1]
 [2 3]]
axis=1
 [[0 2]
 [1 3]]
```

per inserire

```
a1 = np.array([10,20,30,40])
np.insert(a1, 3, [33,34,35])
```

```
array([10, 20, 30, 33, 34, 35, 40])
```

UFUNC

```
list1 = [1,2,3]
list2 = []

for el in list1:
    list2.append(el ** 2)
print("list:", list2)

arr = np.array(list1)
print("array:", arr ** 2)
```

```
list: [1, 4, 9]
array: [1 4 9]
```

Funzione	Operatore	Azione
np.**add**(a1, a2)	a1 **+** a2	addiziona gli elementi corrispondenti
np.**subtract**(a1, a2)	a1 **-** a2	sottrae gli elementi corrispondenti
np.**multiply**(a1, a2)	a1 ***** a2	moltiplica gli elementi corrispondenti
np.**remainder**(a1, a2)	a1 **%** a2	calcola il resto tra elementi corrispondenti
np.**divide**(a1, a2)	a1 **/** a2	divide tra loro gli elementi corrispondenti
np.**power**(a1, a2)	a1 ****** a2	eleva a potenza gli elementi corrispondenti
np.**equal**(a1, a2)	a1 **==** a2	confronta gli elementi corrispondenti e restituisce un vettore di booleani
np.**greater**(a1, a2)	a1 **>** a2	
np.**less**(a1, a2)	a1 **<** a2	
np.**not_equal**(a1, a2)	a1 **!=** a2	
np.**greater_equal**(a1, a2)	a1 **>=** a2	
np.**less_equal**(a1, a2)	a1 **<=** a2	
np.**rint**(a)		arrotonda ogni elemento
np.**floor**(a)		approssima per difetto ogni elemento
np.**ceil**(a)		approssima per eccesso ogni elemento
np.**trunc**(a)		tronca il valore di ogni elemento
np.**around**(a, decimal=2)		arrotonda al 2^ decimale ogni elemento

```
a = np.array([-1.51, -0.22, 0., 0.27, 1.56])

print("rint: ",np.rint(a))
print("floor:",np.floor(a))
print("ceil: ",np.ceil(a))
print("trunc:",np.trunc(a))
print("around",np.around(a,1))
```

```
rint:  [-2. -0.  0.  0.  2.]
floor: [-2. -1.  0.  0.  1.]
ceil:  [-1. -0.  0.  1.  2.]
trunc: [-1. -0.  0.  0.  1.]
around [-1.5 -0.2  0.   0.3  1.6]
```

a.**sum**() o numpy.**sum**(a)	restituisce la somma degli elementi dell'array
a.**prod**() o numpy.**prod**(a)	restituisce il prodotto degli elementi dell'array
np.**sqrt**(a) o numpy.**sqrt**(a)	restituisce la radice quadrata degli elementi dell'array

a.**min**() o numpy.**amin**(a)	restituisce il valore minore tra gli elementi dell'array
a.**max**() o numpy.**amax**(a)	restituisce il valore maggiore tra gli elementi dell'array
a.**argmin**() o numpy.**argmin**(a)	restituisce l'indice del valore minore tra gli elementi (in caso di parimerito restituisce l'indice minore)
a.**argmax**() o numpy.**argmax**(a)	restituisce l'indice del valore maggiore tra gli elementi (in caso di parimerito restituisce l'indice minore)
a.**mean**() o numpy.**mean**(a)	restituisce la media degli elementi dell'array
a.**var**() o numpy .**var**(a)	restituisce la varianza degli elementi dell'array
a.**std**() o numpy.**std**(a)	restituisce la deviazione standard degli elementi dell'array

```
a = np.array([[1,2],[3,4]])

print("sum(): ", a.sum())
print("prod(axis=1):",a.prod(axis=1))
print("min(axis=0):", a.min(axis=0))
print("max(axis=1):", a.max(axis=1))
print("argmin(axis=0):", a.argmin(axis=1))
print("argmax():", a.argmax())
```

```
sum():  10
prod(axis=1): [ 2 12]
min(axis=0): [1 2]
max(axis=1): [2 4]
argmin(axis=0): [0 0]
argmax(): 3
```

PROVA A CALCOLARE LA VARIANZA e LA DEVIAZIONE STANDARD

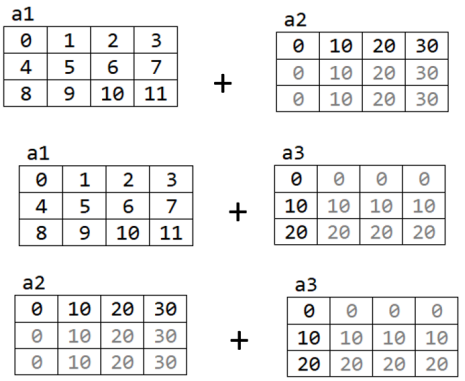
Broadcasting

```
a1 = np.arange(12).reshape(3,4)
a2 = np.array([[0,10,20,30]])   # shape = (1,4)
a3 = np.array([[0],[10],[20]])  # shape = (3,1)
print("a1\n", a1)
print("a2\n", a2)
print("a3\n", a3)
```

```
a1
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
a2
 [[ 0 10 20 30]]
a3
 [[ 0]
 [10]
 [20]]
```

```python
print("shape(3,4) + scalare:\n", a1 + 10)
print("shape(3,4) + shape(1,4):\n", a1 + a2)
print("shape(3,4) + shape(3,1):\n", a1 + a3)
print("shape(1,4) + shape(3,1):\n", a2 + a3)
```

```
shape(3,4) + scalare:
 [[10 11 12 13]
 [14 15 16 17]
 [18 19 20 21]]
shape(3,4) + shape(1,4):
 [[ 0 11 22 33]
 [ 4 15 26 37]
 [ 8 19 30 41]]
shape(3,4) + shape(3,1):
 [[ 0  1  2  3]
 [14 15 16 17]
 [28 29 30 31]]
shape(1,4) + shape(3,1):
 [[ 0 10 20 30]
 [10 20 30 40]
 [20 30 40 50]]
```



Due array sono compatibili per il broadcasting se:

1. hanno la stessa forma
2. hanno lo stesso numero di dimensioni e la 'lunghezza' di ogni dimensione o è uguale o è 1
3. hanno un numero diverso di dimensioni, ma l'array che ne ha meno soddisfa la condizione precedente aggiungendo sufficienti dimensioni di lunghezza 1 a sinistra dello shape di partenza

Quando il broadcasting è possibile, l'array che ha una dimensione = 1, viene modificato in modo che abbia forma compatibile con l'altro e i suoi valori vengono ripetuti lungo quella dimensione e usati per i calcoli. La dimensione dell'array risultato è la maggiore tra le dimensioni degli array di partenza in ciascun asse.