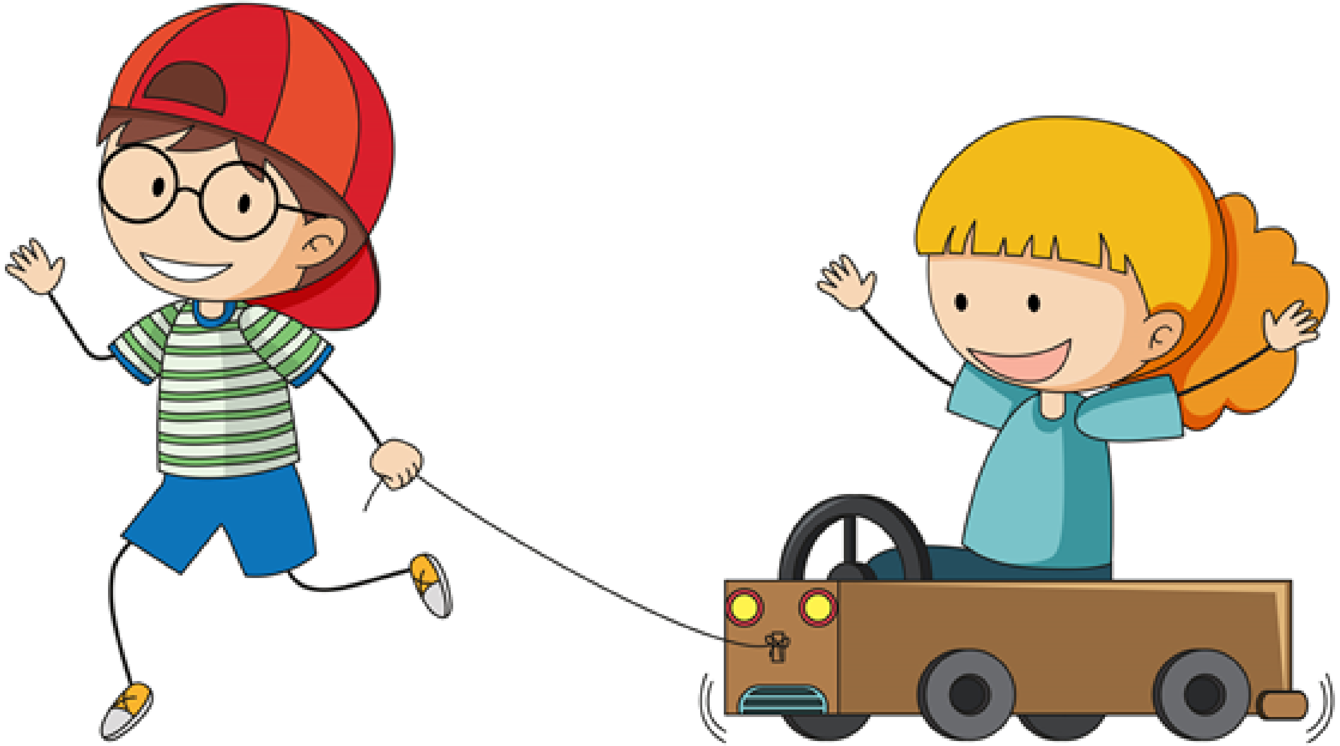
https://graphicmama.com/blog/free-school-clipart-for-your-education-projects/



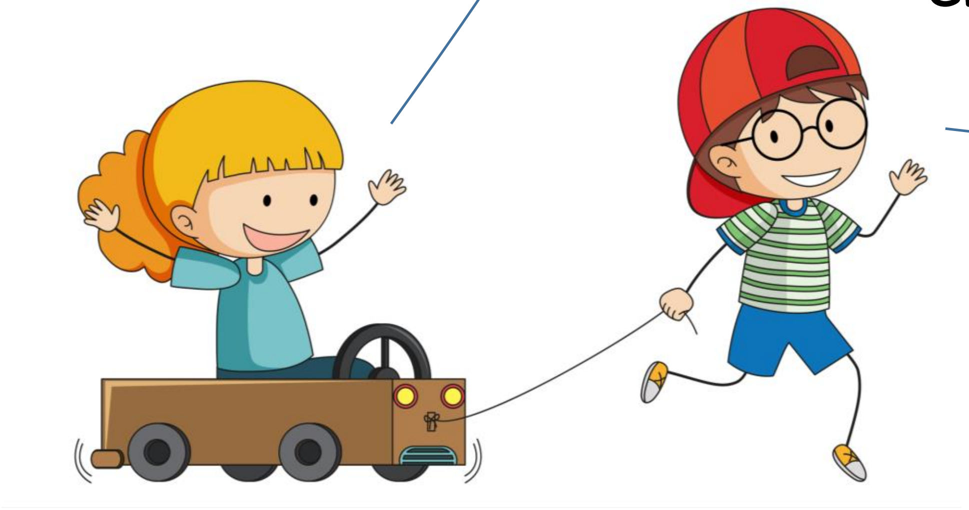
- October 2019
- Dr. Marco A Arocha

L#12 Arrays [2D]

https://docs.python.org/2.5/lib/typesseq.html

¿ Tu crees que podremos salir el domingo a jugar ?
¿ Nos dejara la orden ejecutiva del gobierno ?
Bro’ chequéate el número de la tablilla, por favor

Ya la revise sis’ el último número es primo
y el gobierno dijo pares o impares,
entonces no nos aplica!!



https://graphicmama.com/blog/free-school-clipart-for-your-education-projects/

Creating 2D Arrays

# Statement:
lista =[[5,3,9,7],[7,9,3,5]]
datos = np.array(lista)
# easier:
datos=np.array([[5,3,9,7] , [7,9,3,5]])

axis-1 →

axis-0 ↓

5	3	9	7
7	9	3	5

datos

---

tupla=(2,4)   # (row,col)
unos=np.ones(tupla)
# easier: unos = np.ones((2,4))

1	1	1	1
1	1	1	1

unos

---

ceros =np.zeros((3,3))

Still a tuple
(row,col)

0	0	0
0	0	0
0	0	0

ceros

Indexing: Lists versus Arrays

L is a list; x is an array

L= [[1,2,3], [4,5,**6**], [7,8,9], [10,**11**,12]]

x=np.array([[1,2,3], [4,5,6], [7,8,9], [10,11,12]])

Reference element valued-11:

L[3][1]   # L[row][element within a row]

x[3,1]   # x[row, col]

Reference element valued-6:

L[1][2]

x[1,2]

	[0]	[1]	[2]
L[0]	1	2	3
L[1]	4	5	6
L[2]	7	8	9
L[3]	10	11	12

L

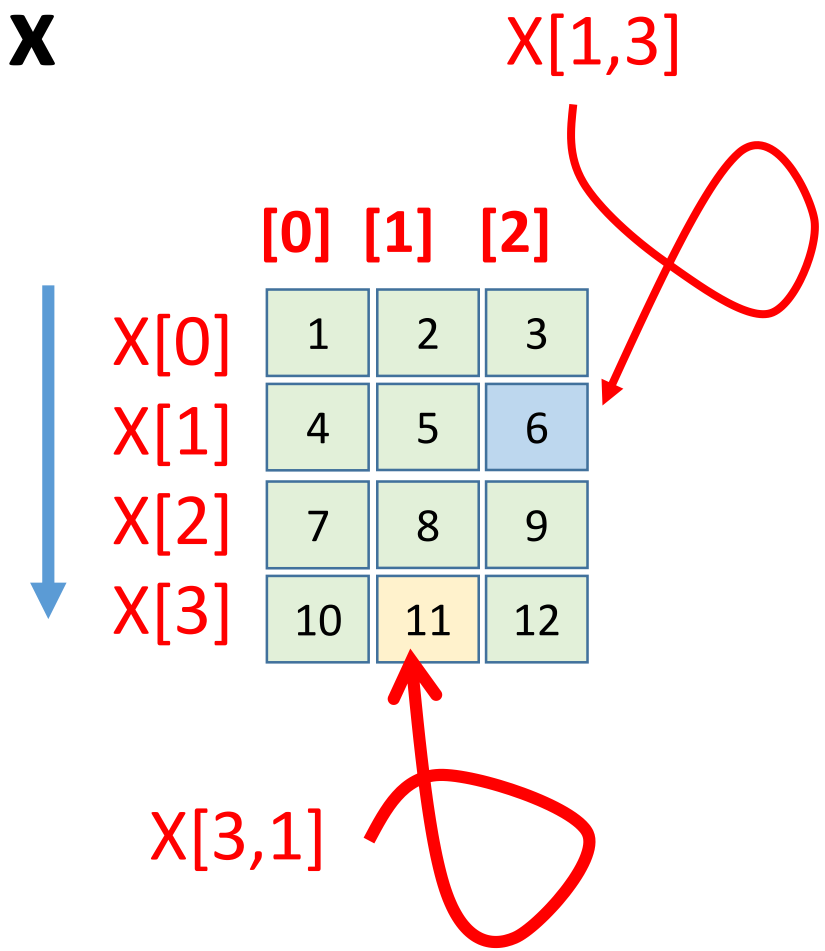
	[,0]	[,1]	[,2]
X[0]	1	2	3
X[1]	4	5	6
X[2]	7	8	9
X[3]	10	11	12

X

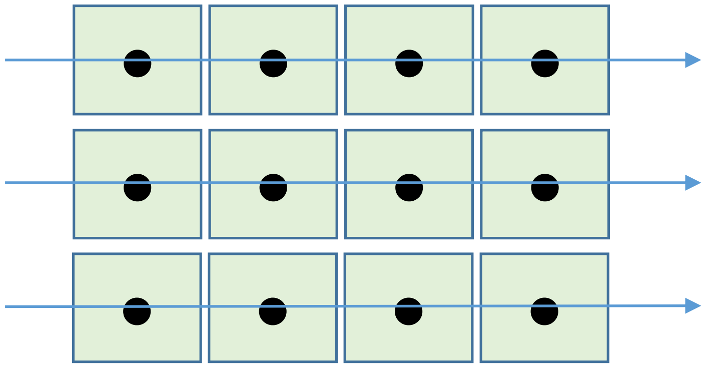
Array construction and storage is row-by-row

Arrays simple operations with single elements (scalar)

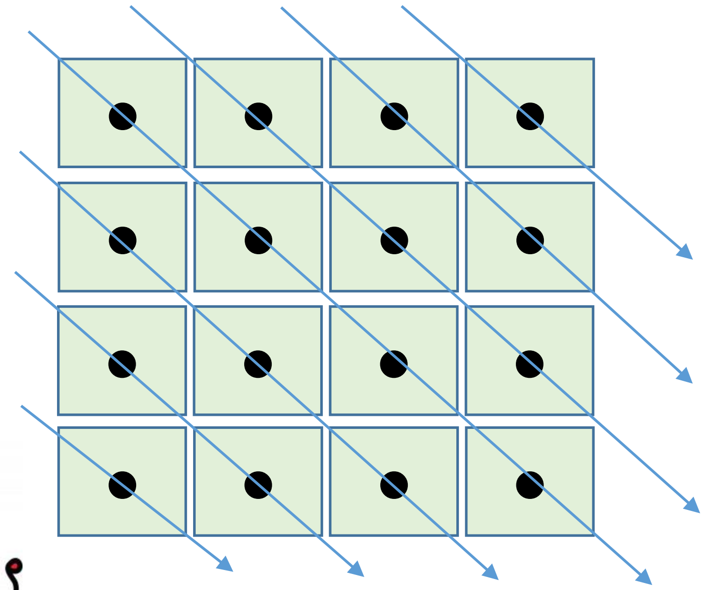
```
# x is an array
x=np.array( [ [1,2,3], [4,5,6], [7,8,9], [10,11,12] ] )

# Print element valued-11:
print(x[3,1])                    # 11

# Multiply 6 and 11:
print(x[1,2]*x[3,1])             # 6*11=66

# Compute sine of element valued-6:
print(np.sin( x[1,2] ) )         # sin(6)=0.104528
```

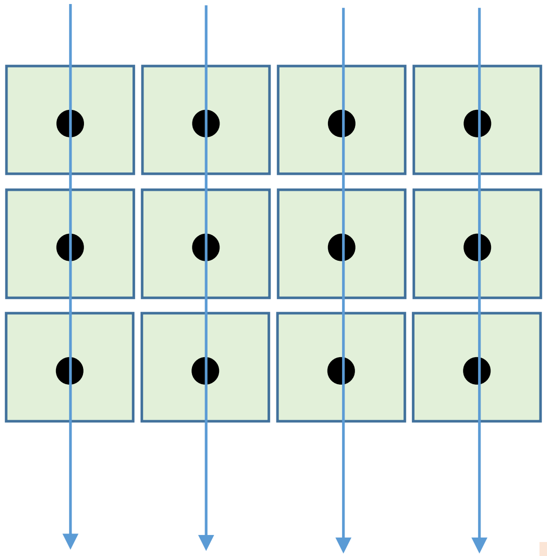


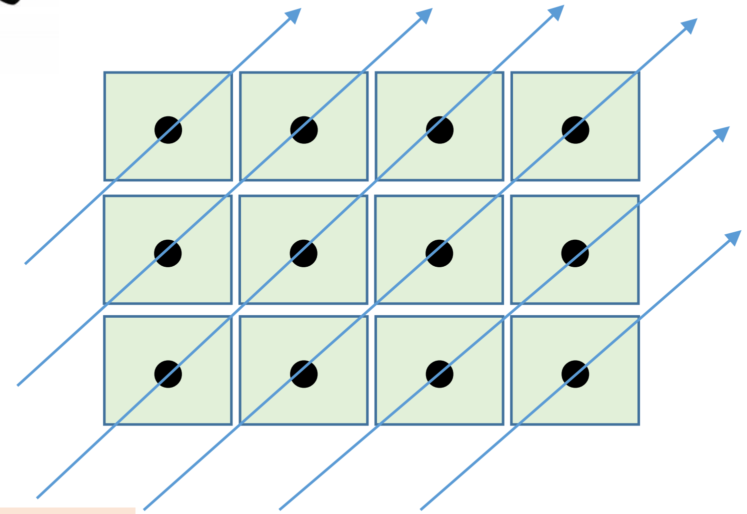
Array Iteration (AI)



Affect all elements, one-by-one in different directions



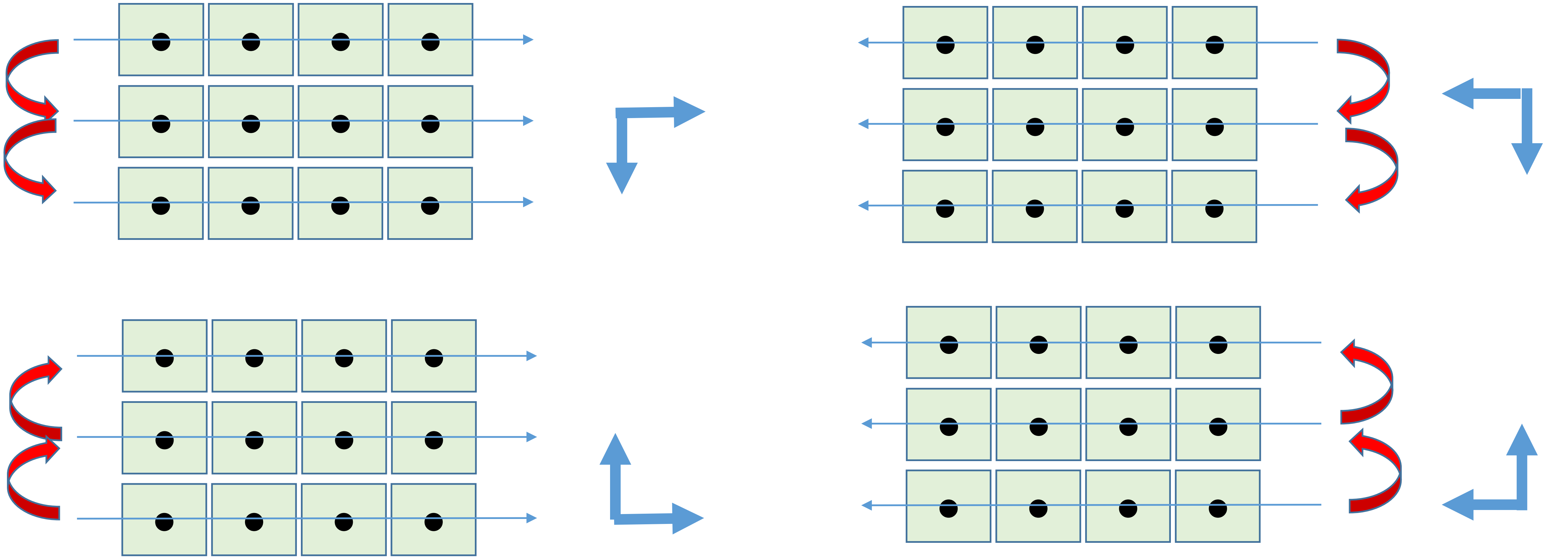




Iterating [“Traveling”] through 2D Arrays

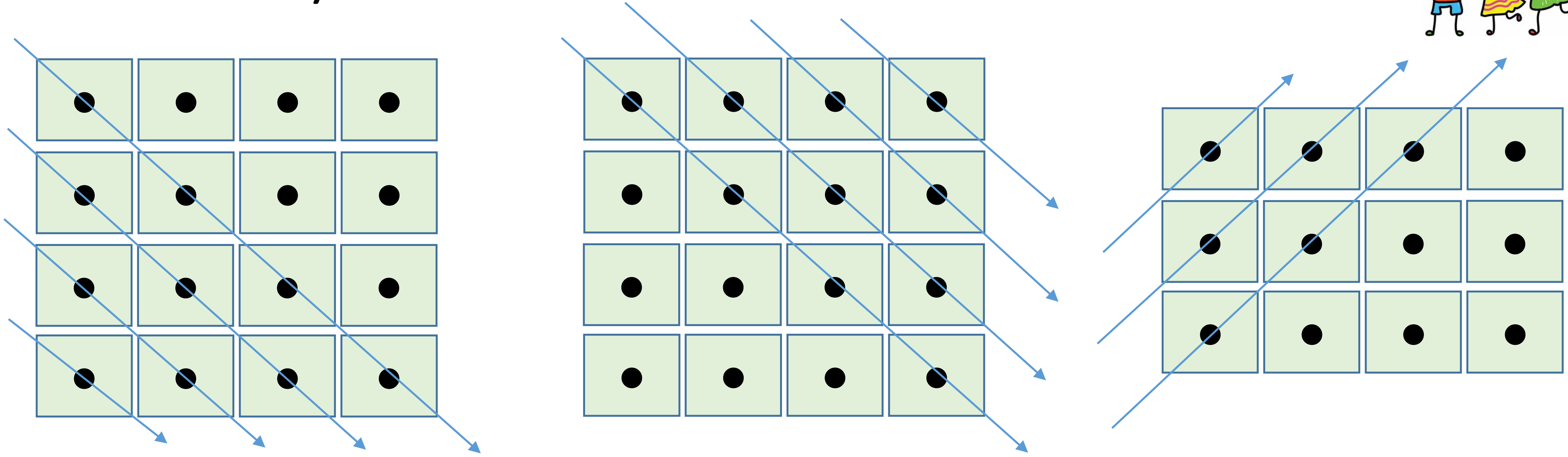
Array Iteration (AI)

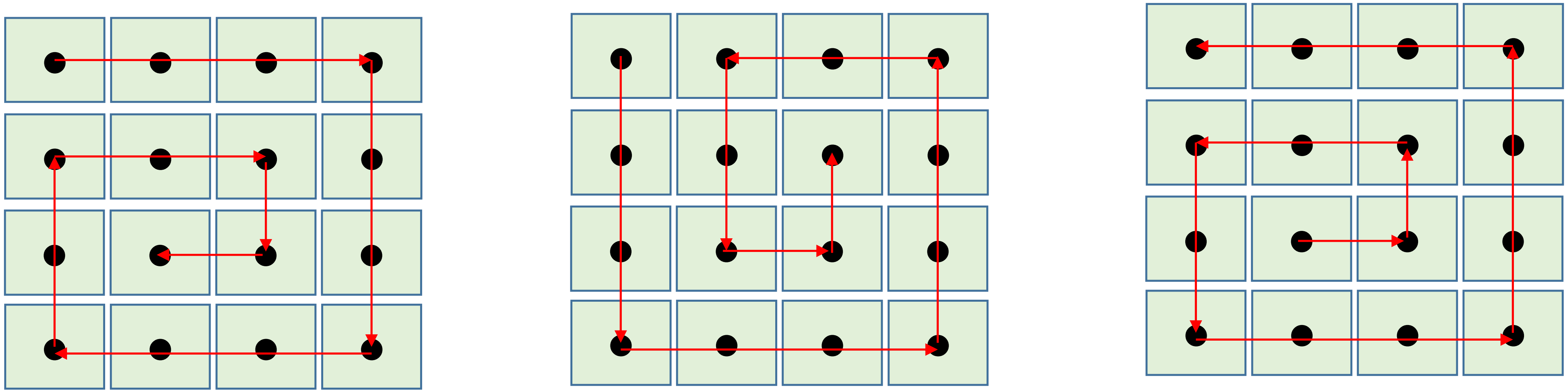
Traveling row-wise



Array Iteration

Affect partial elements

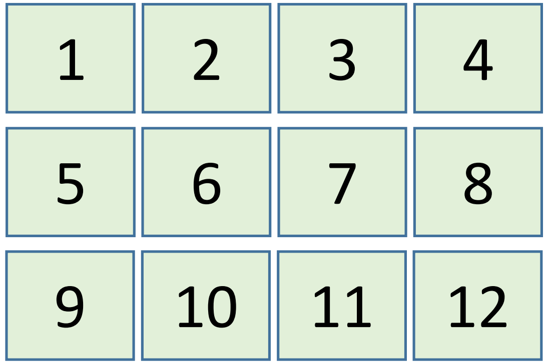




Spiral effect

AI: row-wise

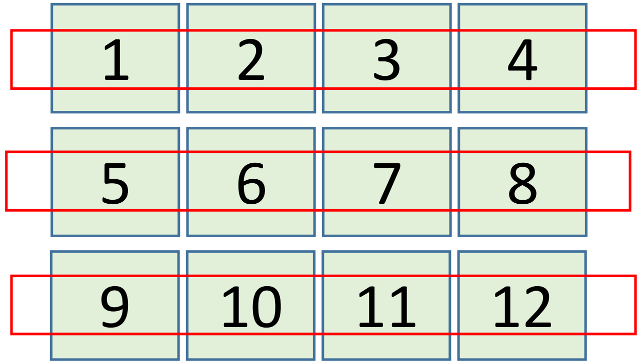
```
X=np.array([[1,2,3,4],
            [5,6,7,8],
            [9,10,11,12]])
print(X)
```



X

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
```

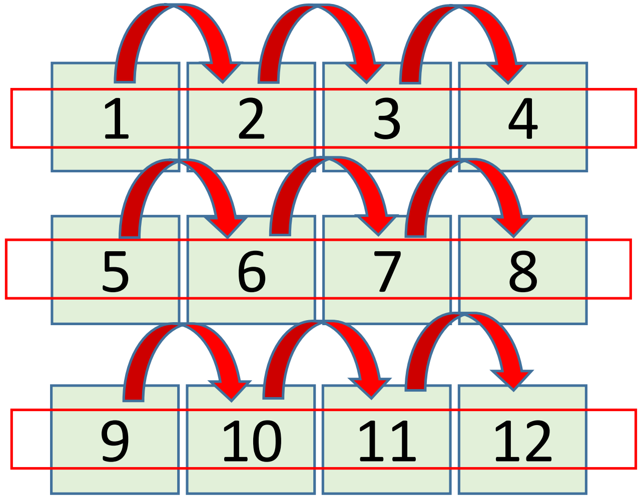
```
for row in X:
    print(row)
```




```
[1 2 3 4]
[5 6 7 8]
[ 9 10 11 12]
```

FILE: iterationArrays01.py

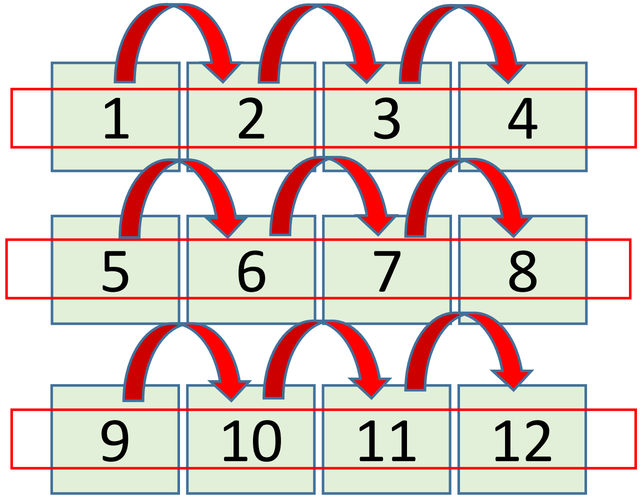
```
for row in X:
    for val in row:
        print(val, end=' ' )
    print()
```




```
1 2 3 4
5 6 7 8
9 10 11 12
```

AI: row-wise processing

```
for row in X:
    for val in row:
        print(val, end=' ' )
    print()
```




```
1  2  3  4
5  6  7  8
9  10  11  12
```

```python
# Sum over each element:
S=0
for row in X:
    for val in row:
        S=S+val
print("Sum is",S)  # Sum is 78
```

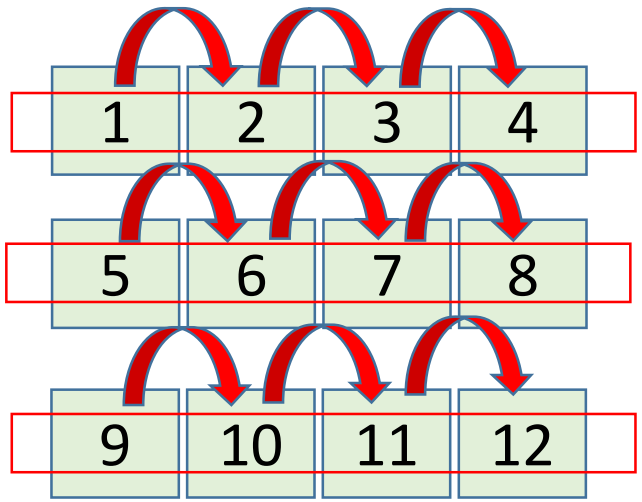
```python
# Even elements:
even=np.array([])
for row in X:
    for val in row:
        if val%2==0:
            even=np.append(even,val)
print("Even are",even)
# Even are [ 2.   4.   6.   8. 10. 12.]
```

FILE: iterationArrays01.py

AI, row-wise: values vs indexes

FILE: iterationArrays01.py

```
# By values:
for row in X:
    for val in row:
        print(val, end=‘ ')
    print()
```



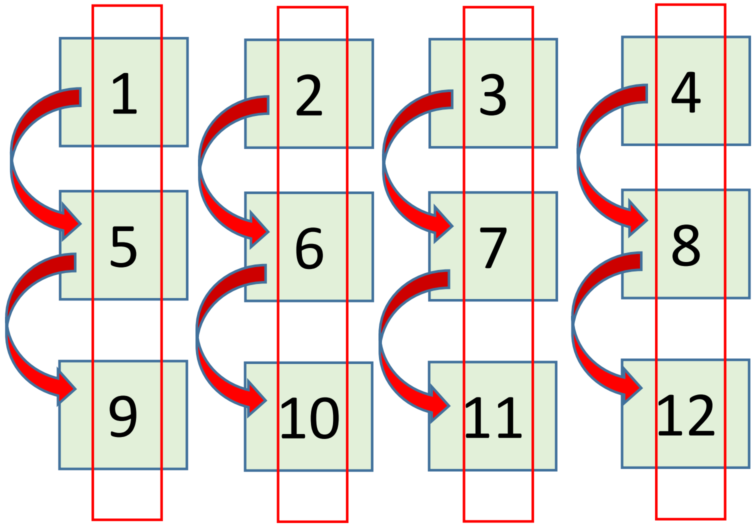

```
# By indexes:
m,n = np.shape(X)
for i in range(m):
    for j in range(n):
        print(X[i,j],end=',')
    print()
```

```
# traveling row-by-row by indexes and List Comprehension
m,n=np.shape(X)
XXX=np.array([[X[i,j] for j in range(n)] for i in range(m)])
print(XXX)
```

AI, column-wise: values vs indexes

```
# By values:
for col in X:
    for val in col:
        print(val, end=',')
    print()
```

From previous solution row-wise solution, we switch the nested loops




```
# By indexes:
m,n=np.shape(X)
for j in range(n):
    for i in range(m):
        print(X[i,j],end=',')
    print()
```

```
1,5,9,
2,6,10,
3,7,11,
4,8,12,
```

FILE: iterationArray02.py

AI: column-wise

X

1	2	3	4
5	6	7	8
9	10	11	12

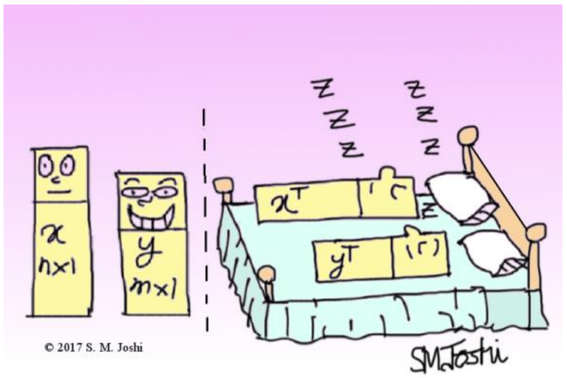
```
X=np.array([[1,2,3,4],
            [5,6,7,8],
            [9,10,11,12]])
```

transposed

transposed

```
for col in X.T:
    print(col)
```

1	5	9
2	6	10
3	7	11
4	8	12



x and y transpose themselves to get some z's

```
for col in X.T:
    for val in col:
        print(val, end=',')
    print()
```

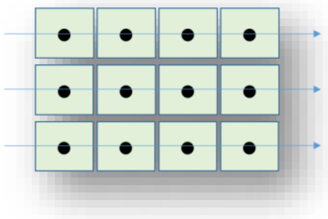
1	5	9
2	6	10
3	7	11
4	8	12

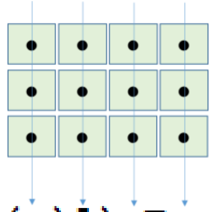
FILE: iterationArrays02.py

AI: By List Comprehension

FILE: Arrays08.py

```
6    import numpy as np
7
8    X=np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
9    print(X)
10
11   # traveling row-by-row by values
12   XX=np.array([[val for val in x_row] for x_row in X])
13   print(XX)
14
15   # traveling row-by-row by indexes
16   m,n=np.shape(X)
17   XXX=np.array([[X[i,j] for j in range(n)] for i in range(m)])
18   print(XXX)
19
20   # traveling column by column by indexes
21   XXXX=np.array([[X[i,j] for i in range(m)] for j in range(n)])
22   print(XXXX)
23   print(XXXX.T)
24
25   # traveling column by column by indexes
26   XXXXX=np.array([[X[i,j] for i in range(m)] for j in range(n)]).T
27   print(XXXXX)
```





9
```
[[ 1 2 3 4]
 [ 5 6 7 8]
 [ 9 10 11 12]]
```

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

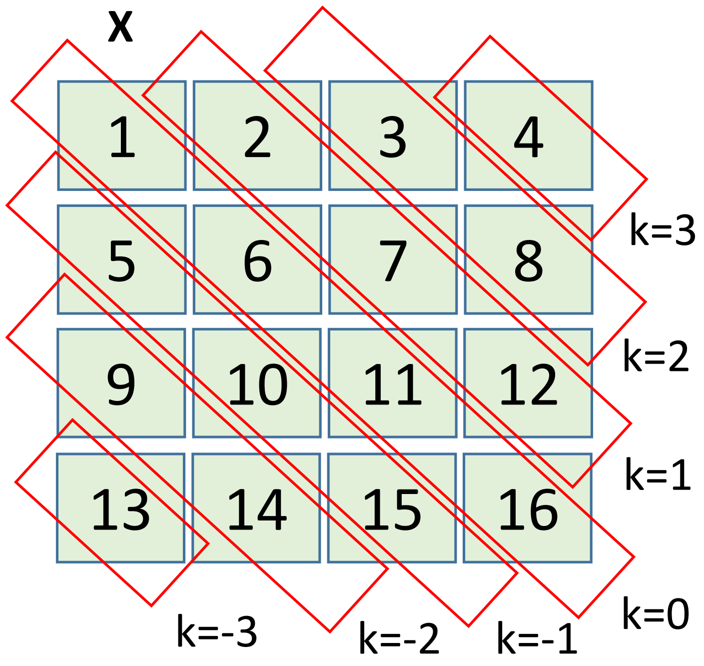
18
```
[[ 1 2 3 4]
 [ 5 6 7 8]
 [ 9 10 11 12]]
```

22
```
[[ 1 5 9]
 [ 2 6 10]
 [ 3 7 11]
 [ 4 8 12]]
```

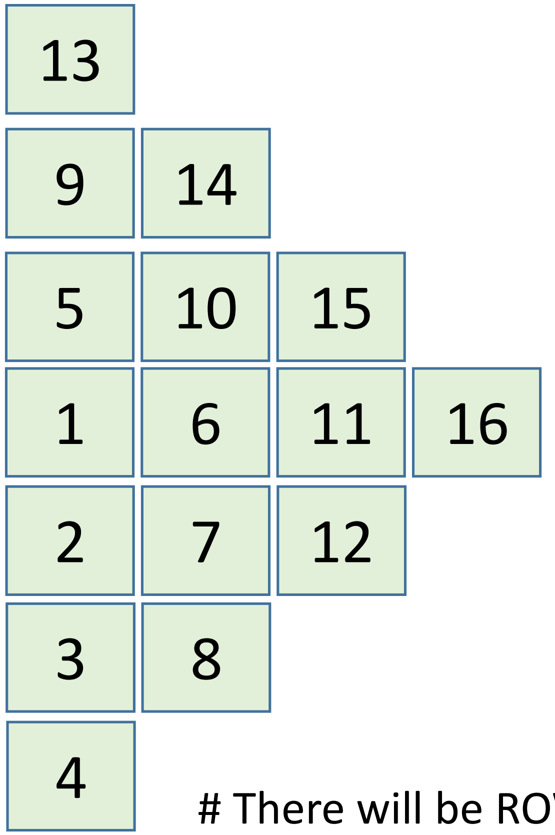
23
```
[[ 1 2 3 4]
 [ 5 6 7 8]
 [ 9 10 11 12]]
```

27
```
[[ 1 2 3 4]
 [ 5 6 7 8]
 [ 9 10 11 12]]
```

AI: diagonal wise




```
X=np.array([[1,2,3,4],
            [5,6,7,8],
            [9,10,11,12]])
```



m=rows
n=cols

```
m,n=np.shape(X)
for k in range(-(m-1),(n-1)+1,1):
    print(np.diag(X,k))
```

There will be ROW+COL-1 lines in the output. i.e., k values
4 rows + 4 columns-1 = 7 (k values)

https://www.geeksforgeeks.org/zigzag-or-diagonal-traversal-of-matrix/

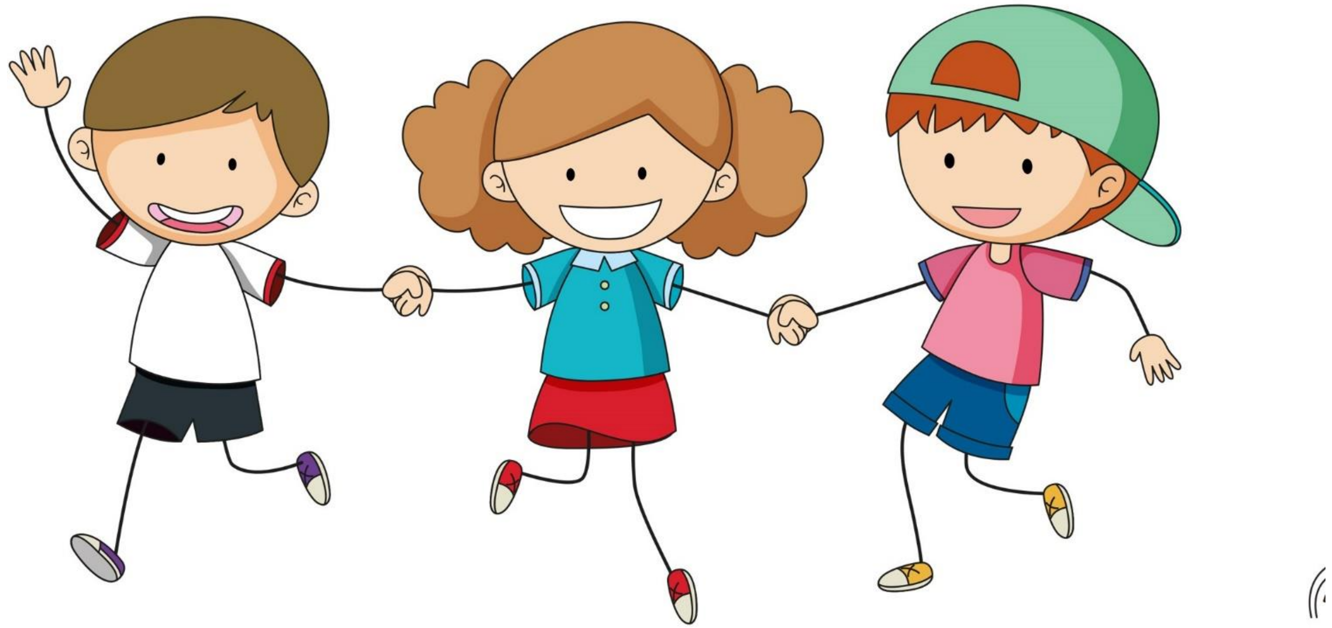
FILE: iterationArrays04.py

Indexing & Slicing

- # Data for next slide:
- x=np.array([[1,2,3], [4,5,6], [7,8,9], [10,11,12]])

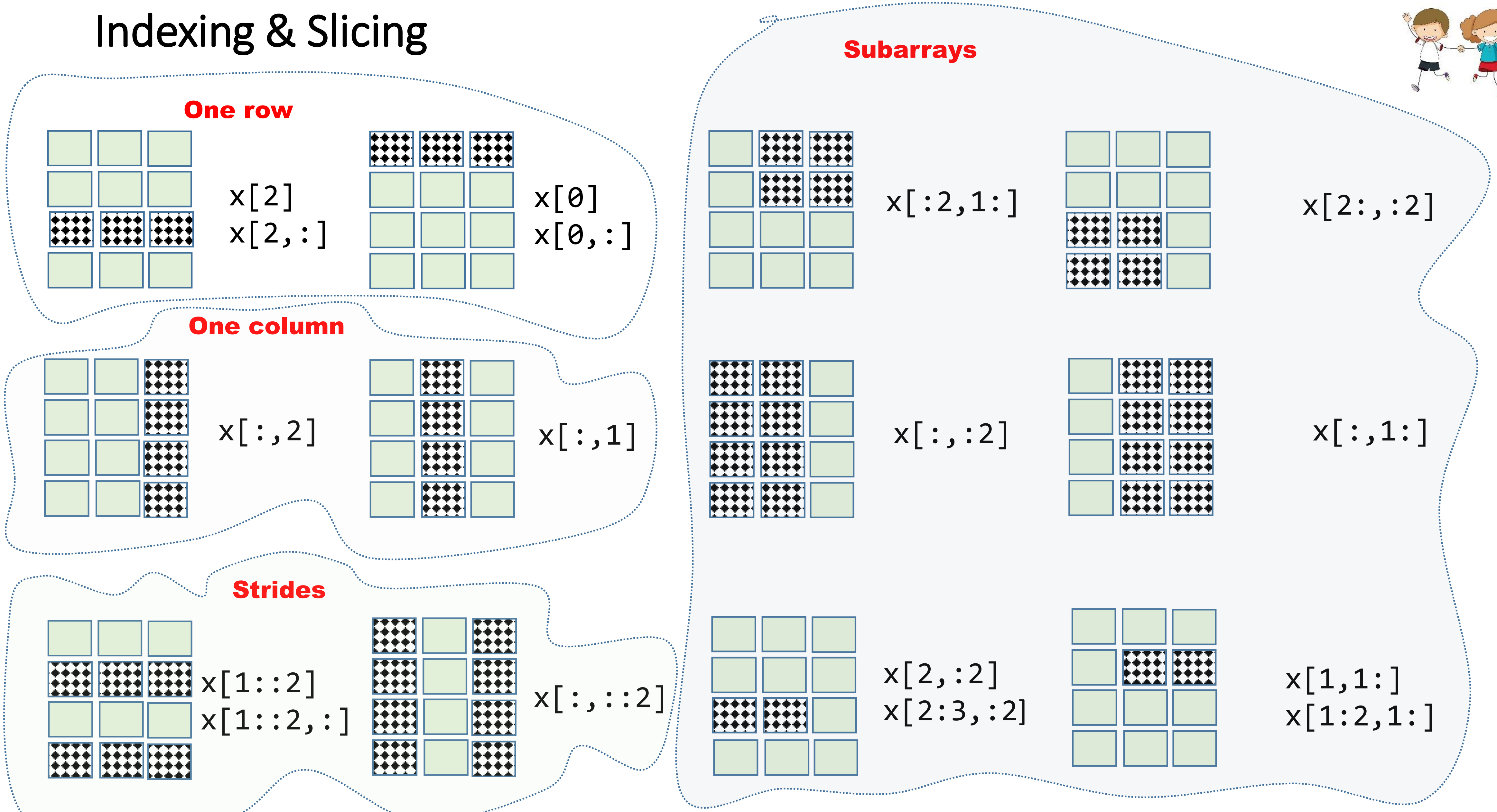
x

1	2	3
4	5	6
7	8	9
10	11	12



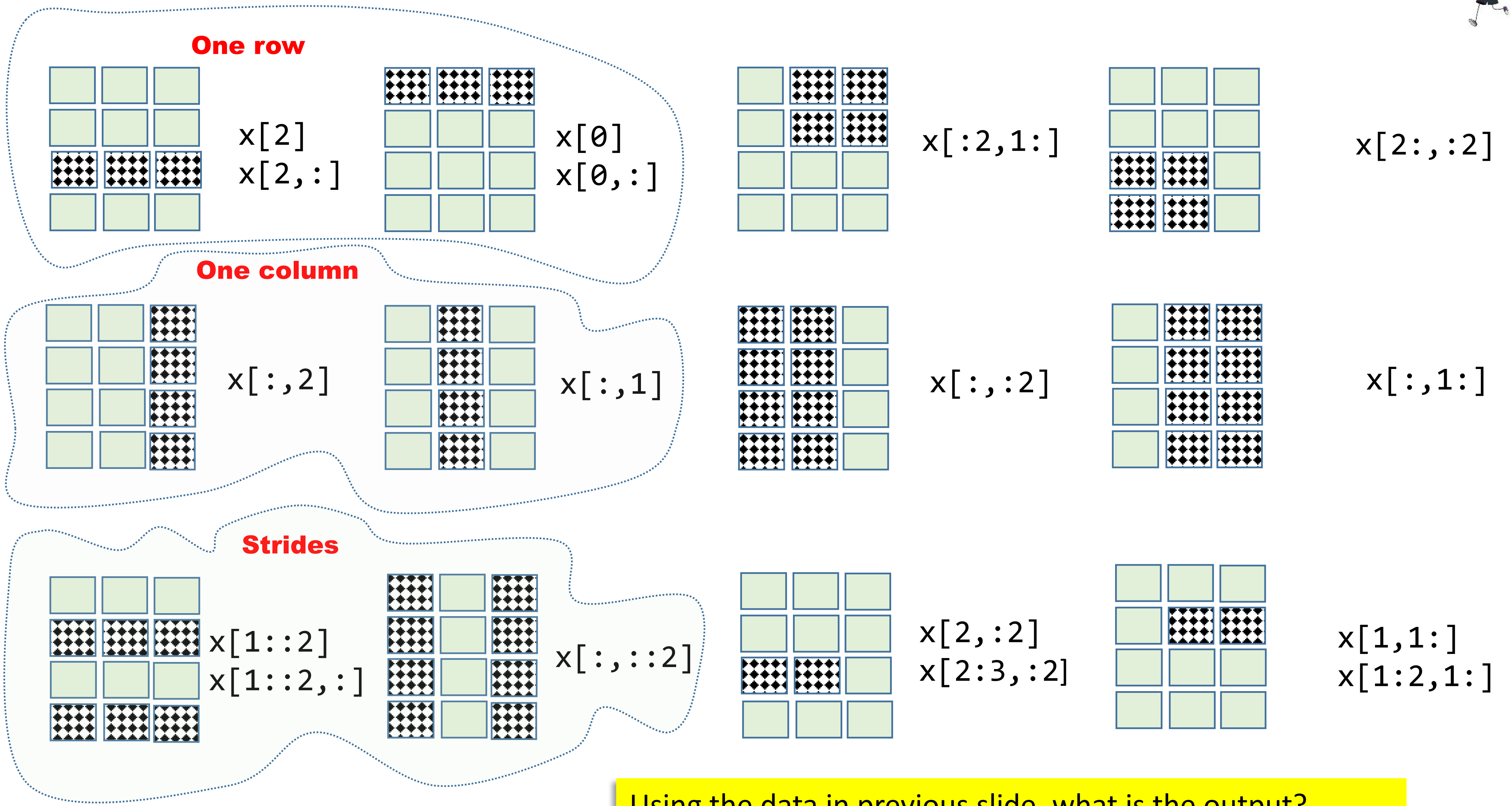
Cartoon from: https://es.vecteezy.com/arte-vectorial/360345-conjunto-de-ninos-del-doodle

Indexing & Slicing



Using the data in previous slide, what is the output?

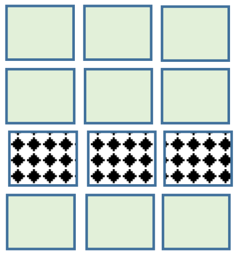
Indexing & Slicing



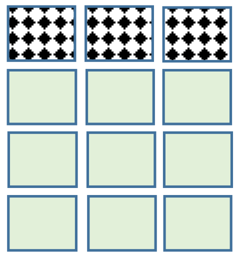
Using the data in previous slide, what is the output?

Indexing & Slicing

One row



```
x[2]
x[2,:]
```



```
x[0]
x[0,:]
```

?

axis-0 = row

x[index]➔x[2]

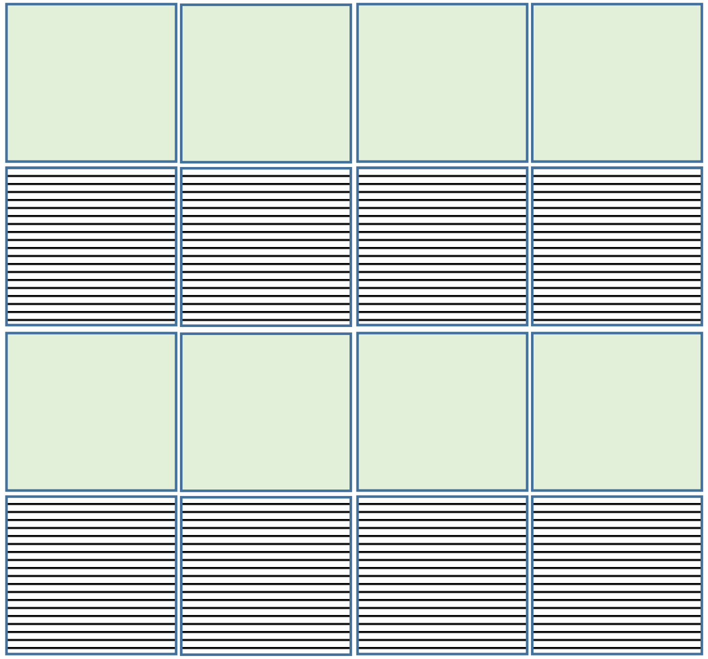
Start:Stop:Step

x[index,index]➔x[2,:]

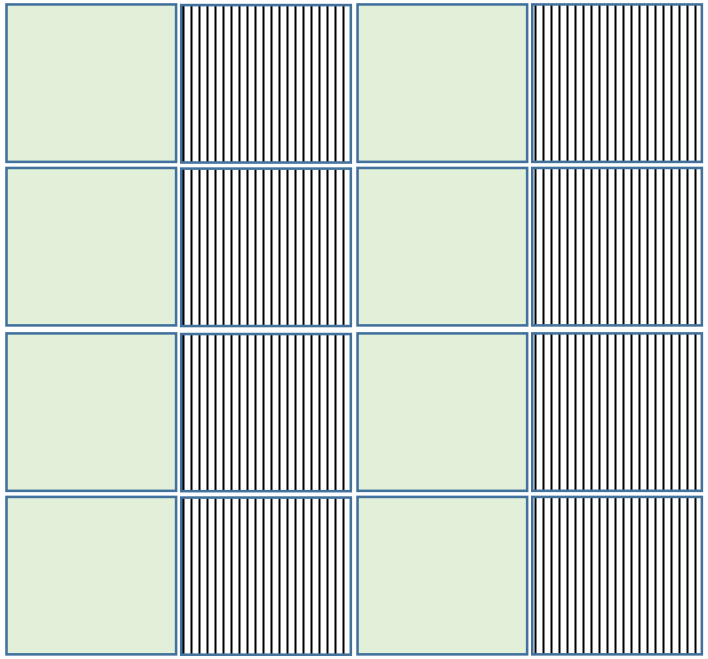
axis-0 = row
axis-1 = col

Using the data in previous slide, what is the output?

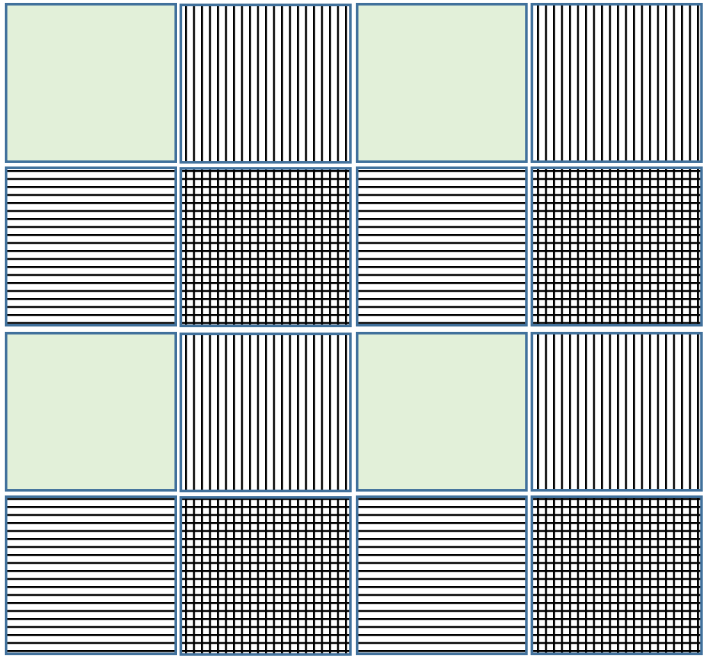
Indexing & Slicing: Demystified



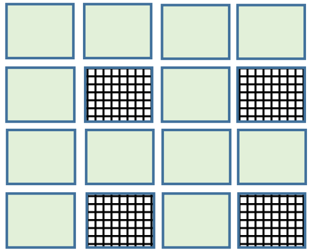
x[1::2, pending]



x[pending, 1::2]



x[1::2,1::2]



Using the data in previous slide, what is the output?

Processing a Slice

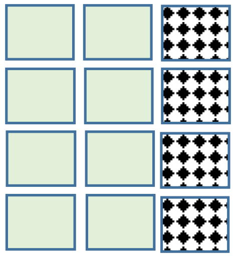
1	3	13	7	9	11	5
2	4	14	8	10	12	6
7	9	19	13	15	17	11
4	6	16	10	12	14	8
5	7	17	11	13	15	9
6	8	18	12	14	16	10
3	5	15	9	11	13	7
8	10	20	14	16	18	12

Given X with all elements, sum all orange elements:

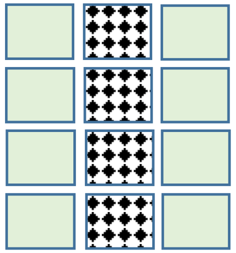
```
S=0
for item x[0::2,0::2]:
     S=S+item
print("Sum is", S)
```

Indexing & Slicing

One column



x[:,2]



x[:,1]

Start:Stop:Step

x[index,index]➔x[: , 2]➔x[0:5:1,2]

axis-0 = row

axis-1 = col

Array Slicing and Gluing pieces back

X=np.array([[1,2,3],[4,5,6]])

1	2	3
4	5	6

2D to 1D

makes a row

A=X[:,0]

1	4

B=X[:,1]

2	5

C=X[:,2]

3	6

Slicing columnwise makes the original slice loose its shape

Array Slicing and Gluing pieces back

Slicing columnwise makes the original slice loose its shape

X=np.array([[1,2,3],[4,5,6]])

1	2	3
4	5	6

2D to 1D

makes a row

A=X[:,0]

1	4

B=X[:,1]

2	5

C=X[:,2]

3	6

1D to 2D

makes a column vector

AA=A[:,np.newaxis]

1
4

BB=B[:,np.newaxis]

2
5

CC=C[:,np.newaxis]

3
6

YY=np.concatenate((AA,BB,CC),axis=1)

```
AA=
 [[1]
 [4]]
BB=
 [[2]
 [5]]
CC=
 [[3]
 [6]]
```

1	2	3
4	5	6

2D to 2D

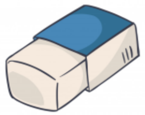
Array Slicing and Gluing pieces back

Slicing columnwise makes the original slice loose its shape
FILE: ArraySlicing01.py

```
import numpy as np; X=np.array([[1,2,3],[4,5,6]])
print("X=\n",X)
A=X[:,0]    # makes a row
AA=A[:,np.newaxis] # makes a column

B=X[:,1]    # makes a row
BB=B[:,np.newaxis]  # makes a column

C=X[:,2]   # makes a row
CC=C[:,np.newaxis]  # makes a column

print("AA=\n",AA); print("BB=\n",BB); print("CC=\n",CC)

print("Concatenate them along axis=1:")
YY=np.concatenate((AA,BB,CC),axis=1); print(YY)

print("Concatenate them along axis=0:")
YY=np.concatenate((AA,BB,CC),axis=0); print(YY)
```

```
X=
 [[1 2 3]
 [4 5 6]]
AA=
 [[1]
 [4]]
BB=
 [[2]
 [5]]
CC=
 [[3]
 [6]]
Concatenate them along axis=1:
[[1 2 3]
 [4 5 6]]
Concatenate them along axis=0:
[[1]
 [4]
 [2]
 [5]
 [3]
 [6]]
```

Delete parts of an array



FILE: deleteArray01.py

The axis along which to delete the subarray defined by indexes. If axis is None, indexes are applied to the flattened array.

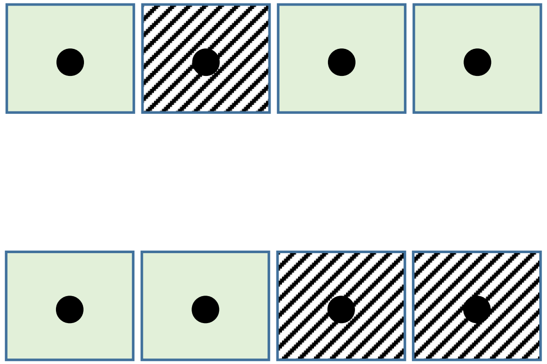
Array name

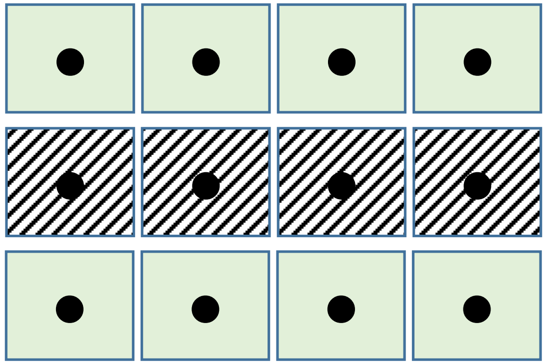
Possibilities:
0/1/None/blank

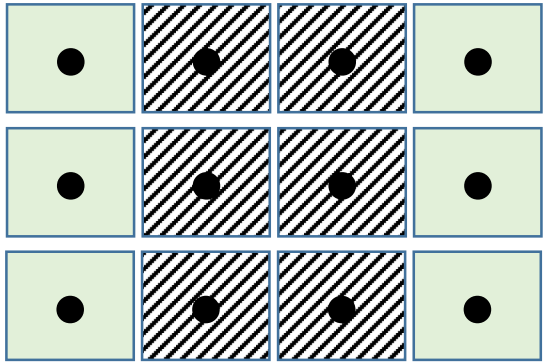
Syntax

numpy.delete(arr, indexes, axis=None)

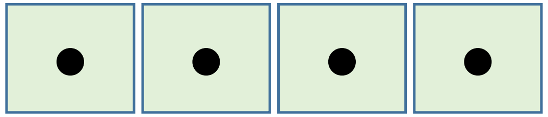
One index or array of indexes along axis to be deleted



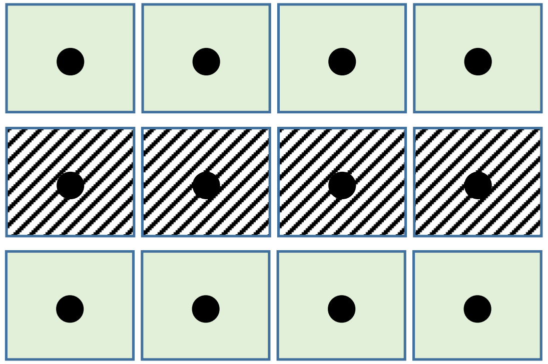




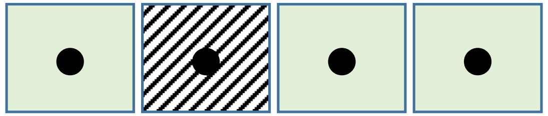
arr1D



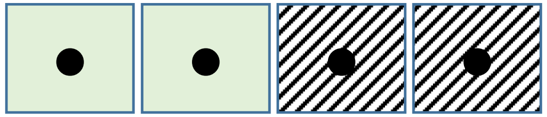
arr2D



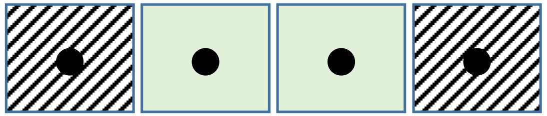
arr1D=np.delete(arr1D,1,axis=0)



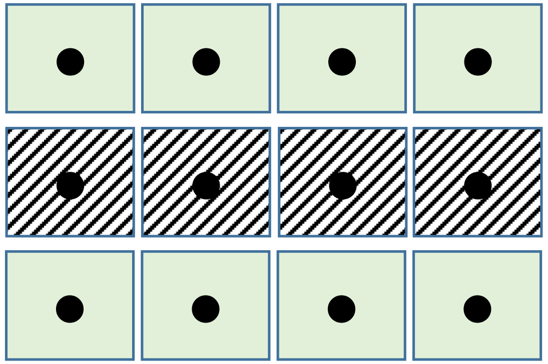
arr1D=np.delete(arr1D,[2,3],axis=0)



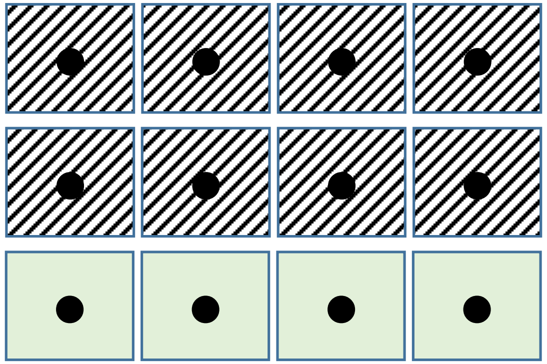
?



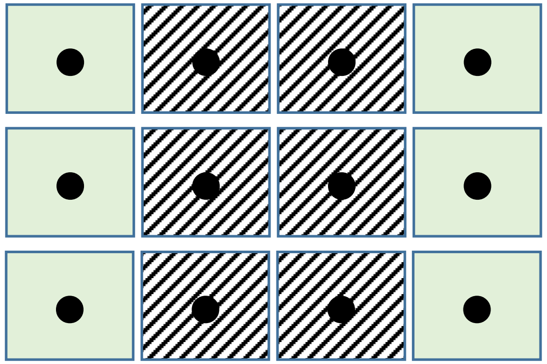
arr2D=np.delete(arr2D, 1, 0)



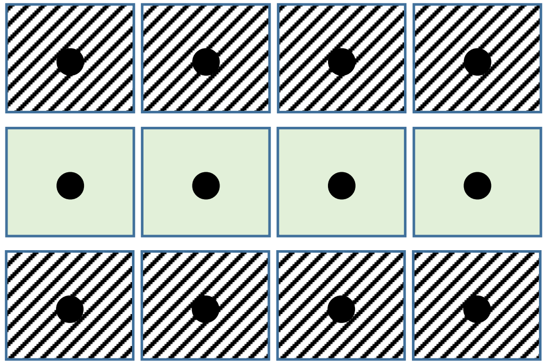
arr2D=np.delete(arr2D, [0,1], 0)



arr2D=np.delete(arr2D, [1,2], 1)

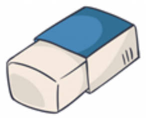


?



FILE: deleteArray01.py

More on Delete: Questions?



```
arr = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
print(arr)
print(np.delete(arr, np.s_[::2], 1))
#delete columns with indexes: 0,2 (step-2)

arr = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
print(np.delete(arr, [1,3,5], None))
#delete indexes: 1,3,5 in a flattened array
```

- OUTPUT

```
[[ 1,  2,  3,  4],
 [ 5,  6,  7,  8],
 [ 9, 10, 11, 12]]

[[ 2,  4],
 [ 6,  8],
 [10, 12]]

[ 1,  3,  5,  7,  8,  9, 10, 11, 12]
```

https://docs.scipy.org/doc/numpy/reference/generated/numpy.s_.html

numpy.s_[: : 2] # a way to build index tuples. In this example indexes are: 0,2

Copy and Deep Copy: Scalar example

Scalars	**Output**
# Example 1	
a=5	
b=a	
b=b+1	
print("a = ", a)	a= 5
print("b = ", b)	b= 6
# Example 2	
a=5	
b=a	
a=a+1	
print("a=", a)	a= 6
print("b=", b)	b= 5

Nothing to worry about with scalars. Copying happens as expected. This is hard copying.

b is modified but a is not

a is modified but b is not

File: copyDeepCopy.py

Copy and Deep Copy: Array case

Assuming A and B are arrays, we can have to types of copying:

```
# Shallow copy
B = A

# Deep copy
B = np.copy(A)
```

Copy and Deep Copy. File: copyDeepCopy.py

Arrays

```
# Arrays
A=np.array([[1,2],[3,4]])
print("A={}".format(A))

B=A                    # Copy A
B[0,0]=5
print("B={}".format(B))
print("A={}".format(A))

print("-----------------------------------")
A=np.array([[1,2],[3,4]])
B=np.copy(A)      # Deep copy
B[0,0]=5
print("B={}".format(B))
print("A={}".format(A))
```

Output

```
A = [ [1 2]
      [3 4] ]

B = [ [5 2]
      [3 4] ]

A = [ [5 2]
      [3 4] ]
-----------------------------------
B = [ [5 2]
      [3 4] ]

A = [ [1 2]
      [3 4] ]
```