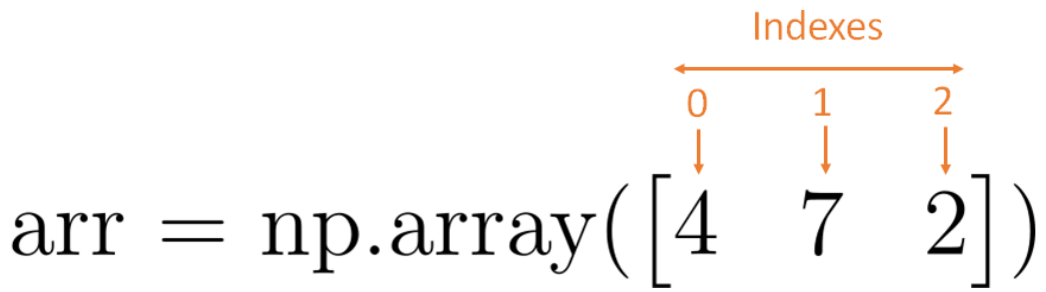
Indexing

- **Selection of an element from the array using its index position is called as indexing.**
- In other words, contents of ndarray object can be accessed by indexing.

Indexing a One-dimensional Array

- The index of a one dimensional numpy array starts at 0.
- This means that the first element would have the index 0.
- The second element would have the index 1, the third element the index 2 and so on.



- To print the number 7 (which is the second element) we get it by indexing the array “arr” with a 1 in square brackets.

$$\text{print}(\text{arr}[1])$$

```
import numpy as np

arr= np.array([4,7,2])

print(arr)
print("\n")

print(arr[1])
```

```
[4 7 2]


7
```

Example 1:

```
#Create 1D numpy array

import numpy as np

array1 = np.arange(0,10)
print(array1)
print("\n")
```

```
print("\n")

#Indexing
array1[2]
```

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


2
```

Example 2:

```
#Create 1D numpy array

import numpy as np

a1 = np.array([1, 2, 3, 4])

print(a1)
print("\n")

#Indexing
print(a1[0])
print("\n")

print(a1[2])
```

```
[1 2 3 4]


1


3
```

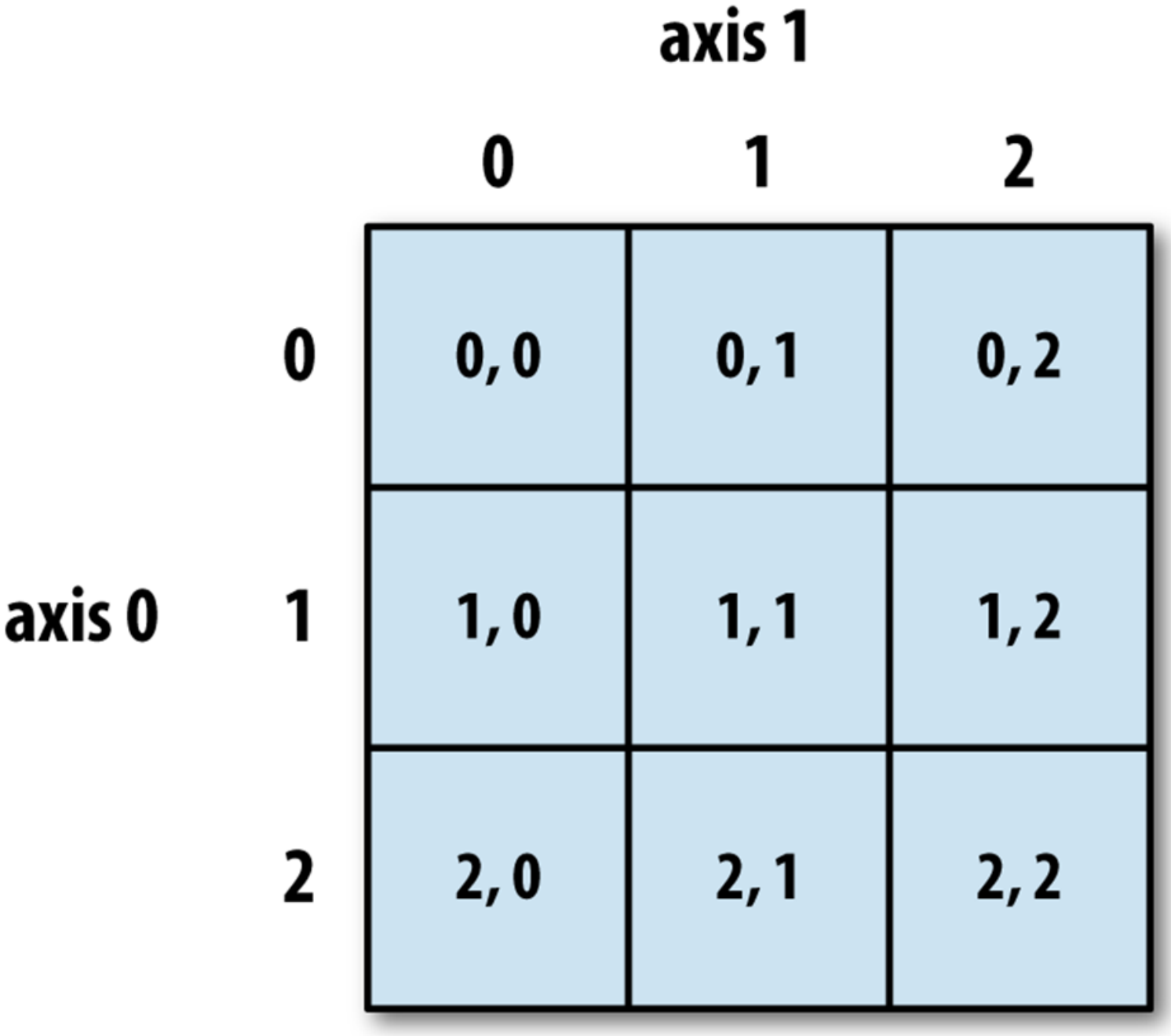
Indexing a Two-dimensional Array

- **To access a single element in 2D array, two indices are used.**
- **One for the row and the other for the column.**
- **Both the column and the row indices start with 0.**

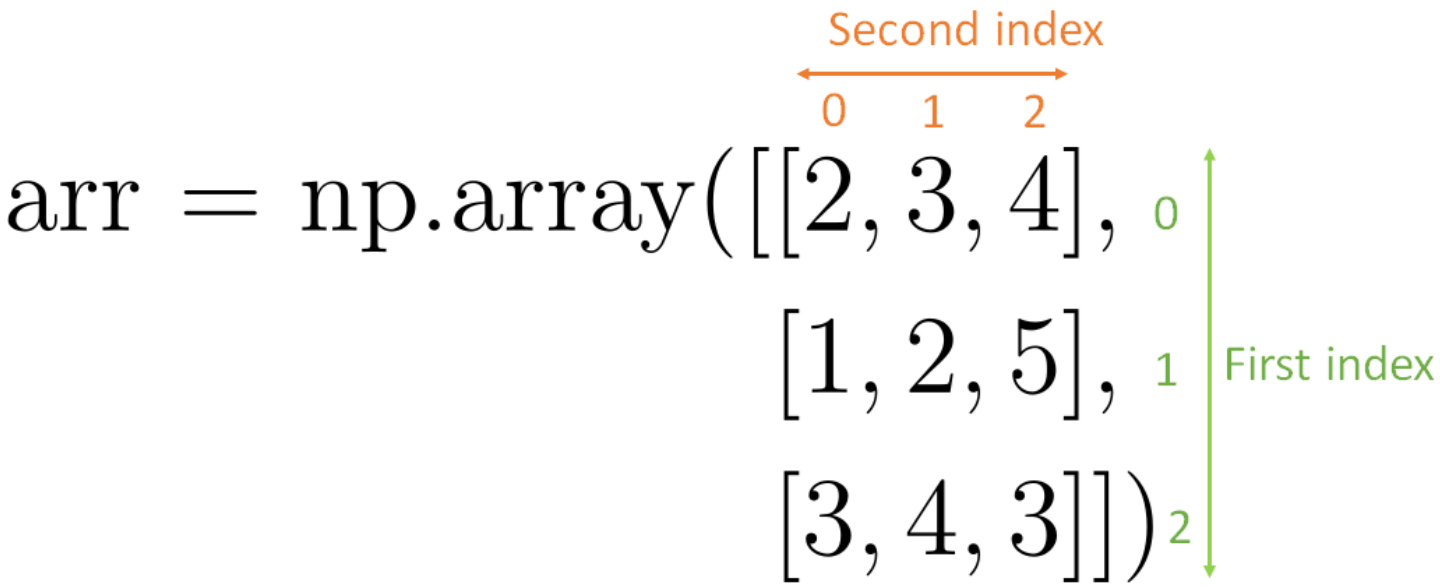
Syntax:

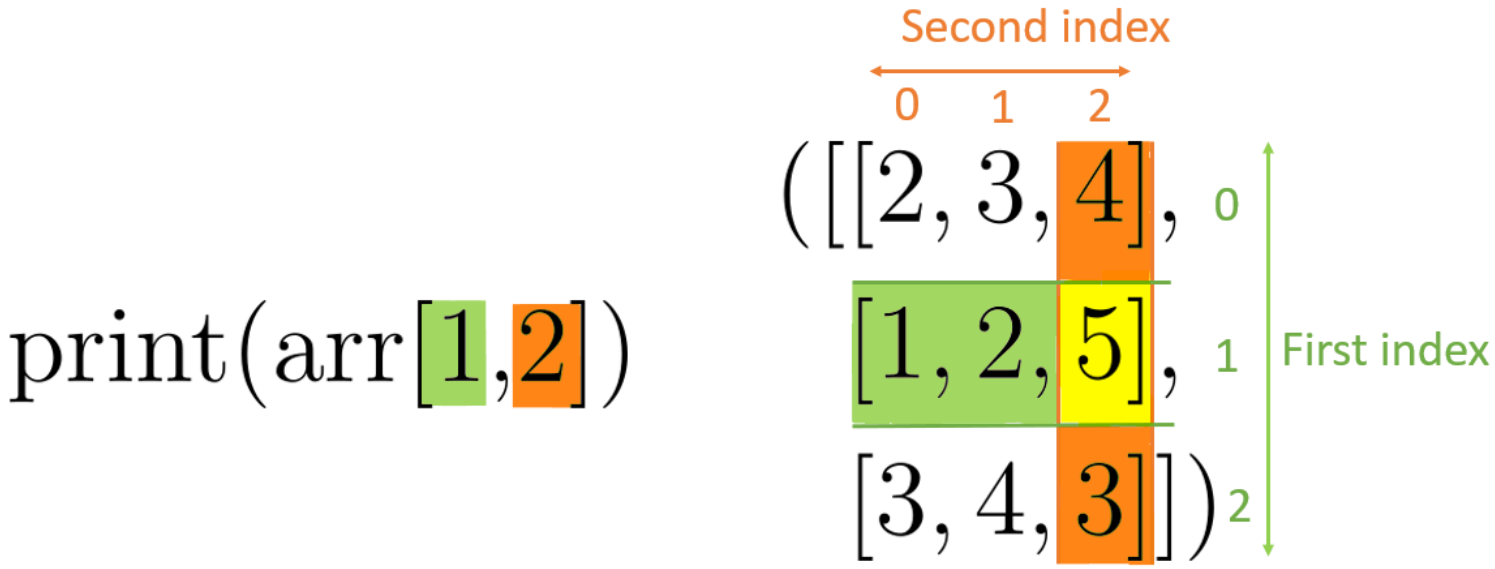
- **The two indices are the i and j values both are placed inside the square brackets, separated by a comma.**
- i selects the row, and j selects the column.

Index Positions of 2D array:



Example 1:





The first index is always along the axis surrounded by the least amount of brackets

```
import numpy as np

arr= np.array([[2,3,4],[1,2,5],[3,4,3]])
print(arr)
print("\n")

print(arr[1,2])
```

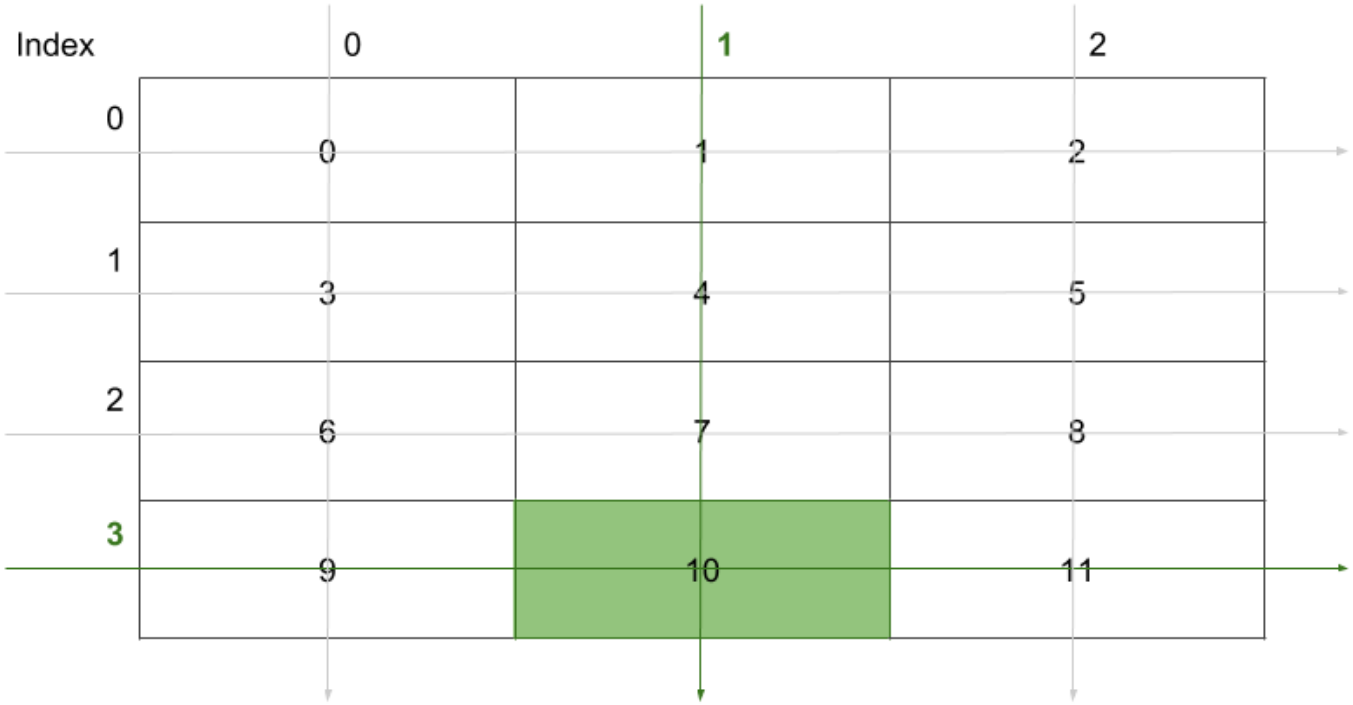
```
[[2 3 4]
 [1 2 5]
 [3 4 3]]


5
```

Example 2:

> To access the value '10,' use the index '3' for the row and index '1' for the column in this example.

```
#Create a 2D array

array1 = np.arange(12).reshape(4,3)
print(array1)
print("\n")

#Indexing
array1[3][1]
```

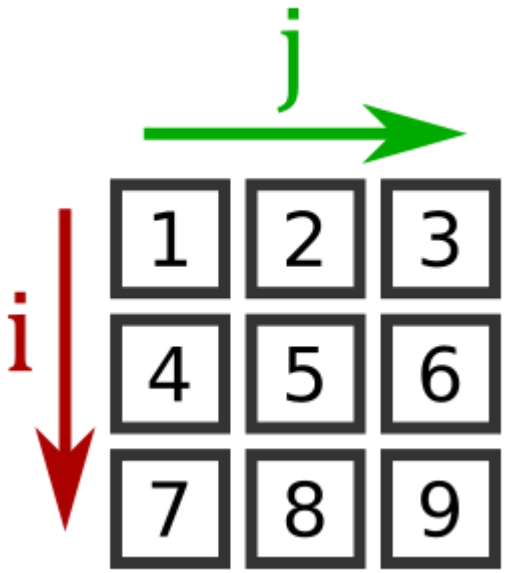
```
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]


10
```



Example 3:

```
#Create a 2D numpy array

import numpy as np

a2 = np.array([[1, 2, 3],
               [4, 5, 6],
               [7, 8, 9]])

print(a2)
print("\n")

#Indexing
print(a2[2, 1])
```

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


8
```



Indexing a Three-dimensional Array

- **To access the elements of three or more dimensional arrays, you have to provide one index for each dimension.**
- For example, to access a three-dimensional array, include the index for the third dimension as well.

(It may be difficult to imagine a three-dimensional array.)

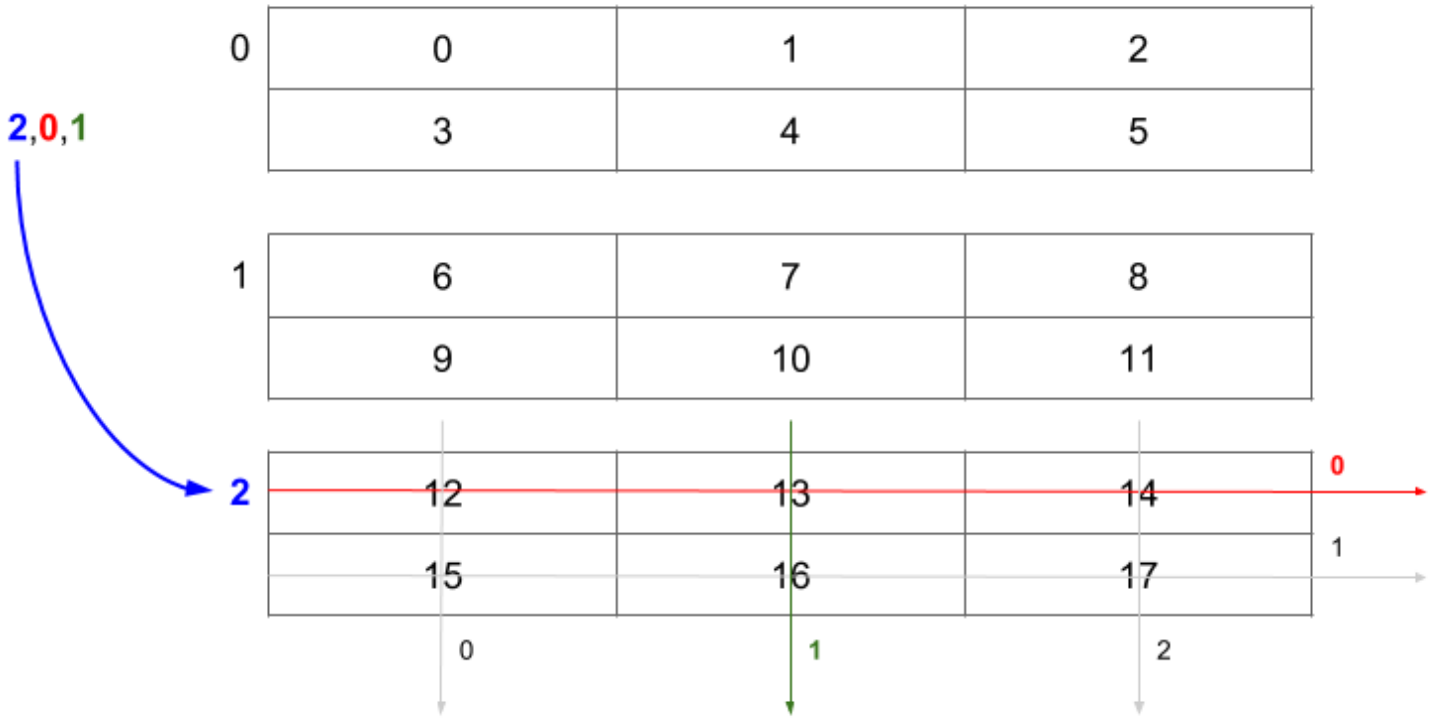
Example 1:

```
#create a three-dimensional array:
array1 = np.arange(18).reshape(3,2,3)

print(array1)
print("\n")


print(array1[2][0][1])
```

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

 [[ 6  7  8]
  [ 9 10 11]]

 [[12 13 14]
  [15 16 17]]]
```

```
13
```

- Note that there are three two-dimensional arrays of size two by three. If you reshape the array into size (5,3,4), there will be five two-dimensional arrays with a size of three by four.
- So, to retrieve the value '13', first go the third two-dimensional array by specifying the index '2.' And once you find the desired two-dimensional array, access the element you need.
- For our case, you need to use the index 2, 0, and 1, where '0' indicates the row 0 and '1' indicates the column 1 within the third two-dimensional array.

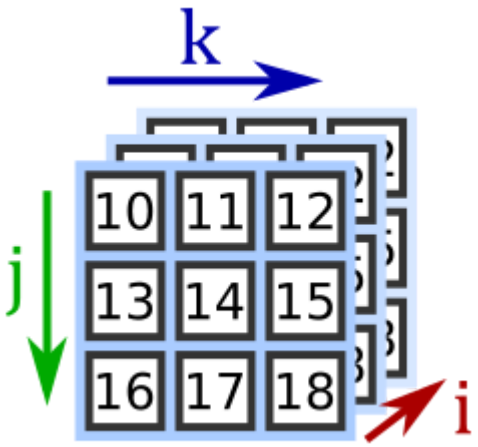


Example 2:

```
#create a 3 dimensional numpy array
import numpy as np

a3 = np.array([[[10, 11, 12], [13, 14, 15], [16, 17, 18]], [[20, 21, 22], [23, 24, 25], [26, 27, 28]

print(a3)
print("\n")

#This selects matrix index 2 (the final matrix), row 0, column 1, giving a value 31.
print(a3[2, 0, 1])
```

```
[[[10 11 12]
  [13 14 15]
  [16 17 18]]

 [[20 21 22]
  [23 24 25]
```

```
 [26 27 28]]

[[30 31 32]
 [33 34 35]
 [36 37 38]]]


31
```



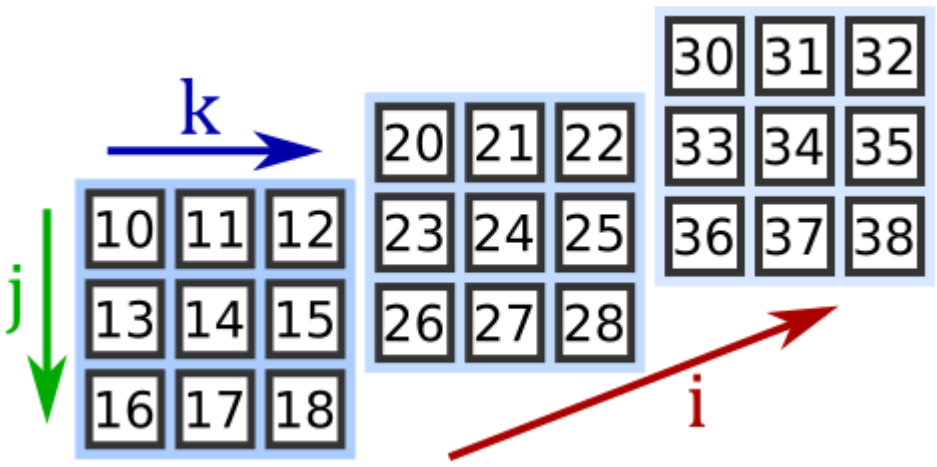
A 3D array is like a stack of matrices:

> **The first index, i, selects the matrix**

> **The second index, j, selects the row**

> **The third index, k, selects the column**

Here is the same diagram, spread out a bit so we can see the values:



Example 3:

```
arr3d = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
print(arr3d)
print("\n")

print(arr3d[0])
print("\n")
```

```
print(arr3d[1, 0])
print("\n")

print(arr3d[1])
print("\n")
```

```
[[[ 1  2  3]
  [ 4  5  6]]

 [[ 7  8  9]
  [10 11 12]]]


[[1 2 3]
 [4 5 6]]


[7 8 9]


[[ 7  8  9]
 [10 11 12]]
```

Example 5:

```
import numpy as np

arr= np.array([[[2,1],[4,5],[6,2]],[[3,4],[7,6],[10,2]]])

print(arr)
print("\n")

print(arr[1,2,0])
```

```
[[[ 2  1]
  [ 4  5]
  [ 6  2]]

 [[ 3  4]
  [ 7  6]
  [10  2]]]


10
```

Boolean Indexing

- **Selection of elements in an array** using the **index values resulted from an expression is called Boolean indexing.**
- It **has some boolean expression as the index.**

- **Those elements are returned which satisfy that Boolean expression.**
- It **is used for filtering the desired element values.**

using > operator

```
import numpy as np

x = np.array([[ 0,  1,  2],[ 3,  4,  5],[ 6,  7,  8],[ 9, 10, 11]])
print('Our array is:' )
print(x)
print('\n')

# Now we will print the items greater than 5
print('The items greater than 5 are:')
print(x[x > 5])
```

```
Our array is:
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]


The items greater than 5 are:
[ 6  7  8  9 10 11]
```

using < or == operator

```
import numpy as np

A = np.array([4, 7, 3, 4, 2, 8])

print(A[A == 4])

print(A[A < 5])
```

```
[4 4]
[4 3 4 2]
```

```
import numpy as np

a = np.array([10, 40, 80, 50, 100])

print(a[a>50])
```

```
[ 80 100]
```

using >= or <= operator

```
B = np.array([[42,56,89,65],
              [99,88,42,12],
              [55,42,17,18]])
```

```
print(B[B>=42])
```

```
[42 56 89 65 99 88 42 55 42]
```

```
C = np.array([123,188,190,99,77,88,100])
A = np.array([4,7,2,8,6,9,5])

R = C[A<=5]
print(R)
```

```
[123 190 100]
```

using % or power (**) operator

```
import numpy as np

a = np.array([10, 40, 80, 50, 100])

print(a[a%40==0]**2)
```

```
[1600 6400]
```

using two conditions with logical or '|'

```
b = np.array([[5, 5],[4, 5],[16, 4]])

print(b[(b%2 == 0) | (b%3 == 0)])
```

```
[ 4 16  4]
```

```
import numpy as np

A = np.array([
[12, 13, 14, 12, 16, 14, 11, 10,  9],
[11, 14, 12, 15, 15, 16, 10, 12, 11],
[10, 12, 12, 15, 14, 16, 10, 12, 12],
[ 9, 11, 16, 15, 14, 16, 15, 12, 10],
[12, 11, 16, 14, 10, 12, 16, 12, 13],
[10, 15, 16, 14, 14, 14, 16, 15, 12],
[13, 17, 14, 10, 14, 11, 14, 15, 10],
[10, 16, 12, 14, 11, 12, 14, 18, 11],
[10, 19, 12, 14, 11, 12, 14, 18, 10],
[14, 22, 17, 19, 16, 17, 18, 17, 13],
[10, 16, 12, 14, 11, 12, 14, 18, 11],
[10, 16, 12, 14, 11, 12, 14, 18, 11],
[10, 19, 12, 14, 11, 12, 14, 18, 10],
[14, 22, 12, 14, 11, 12, 14, 17, 13],
[10, 16, 12, 14, 11, 12, 14, 18, 11]])
```

```
print(A[A>15])
print("\n")

print(A[~(A%3 == 0)])
print("\n")

print(A[(A%2 == 0) & (A%3 == 0)])
```

```
[16 16 16 16 16 16 16 16 16 17 16 18 19 18 22 17 19 16 17 18 17 16 18 16
 18 19 18 22 17 16 18]


[13 14 16 14 11 10 11 14 16 10 11 10 14 16 10 11 16 14 16 10 11 16 14 10
 16 13 10 16 14 14 14 16 13 17 14 10 14 11 14 10 10 16 14 11 14 11 10 19
 14 11 14 10 14 22 17 19 16 17 17 13 10 16 14 11 14 11 10 16 14 11 14 11
 10 19 14 11 14 10 14 22 14 11 14 17 13 10 16 14 11 14 11]


[12 12 12 12 12 12 12 12 12 12 12 12 12 12 18 12 12 18 18 12 12 18 12
 12 18 12 12 18 12 12 12 12 18]
```

```
import numpy as np

a = np.array([1, 2+6j, 5, 3.5+5j, 10, 50])

print(a[np.iscomplex(a)])
```

```
[2. +6.j 3.5+5.j]
```

Question

Extract from the array np.array([3,4,6,10,24,89,45,43,46,99,100]) with Boolean indexing all the number

which are not divisible by 3

which are divisible by 5

which are divisible by 3 and 5

which are divisible by 3 and set them to 42

```
import numpy as np
A = np.array([3,4,6,10,24,89,45,43,46,99,100])

div3 = A[A%3!=0]
print("Elements of A not divisible by 3:")
print(div3)
print("\n")


div5 = A[A%5==0]
print("Elements of A divisible by 5:")
print(div5)
print("\n")
```

```
print("Elements of A, which are divisible by 3 and 5:")
print(A[(A%3==0) & (A%5==0)])
print("\n")
```

```
A[A%3==0] = 42
print("""New values of A after setting the elements of A,
which are divisible by 3, to 42:""")
print(A)
print("\n")
```

```
Elements of A not divisible by 3:
[  4  10  89  43  46 100]


Elements of A divisible by 5:
[ 10  45 100]


Elements of A, which are divisible by 3 and 5:
[45]


New values of A after setting the elements of A,
which are divisible by 3, to 42:
[ 42   4  42  10  42  89  42  43  46  42 100]
```

```
```

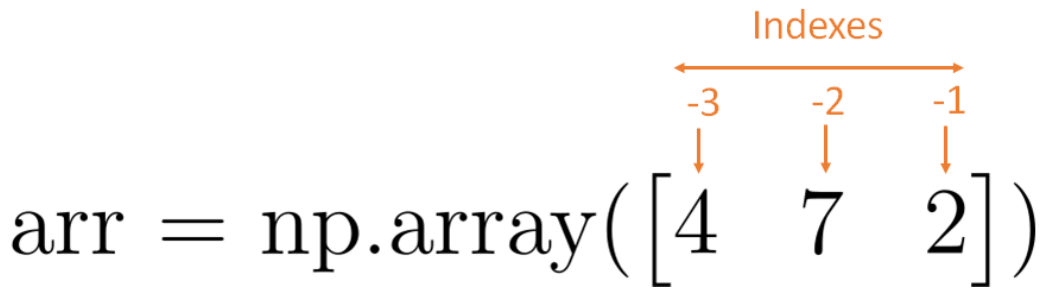
```
```

Additional Points

Negative indexing

- **We can provide negative Indexes to count backwards in the array.**
- **The index of the last index is -1, the index of the second last element is -2 and so on.**

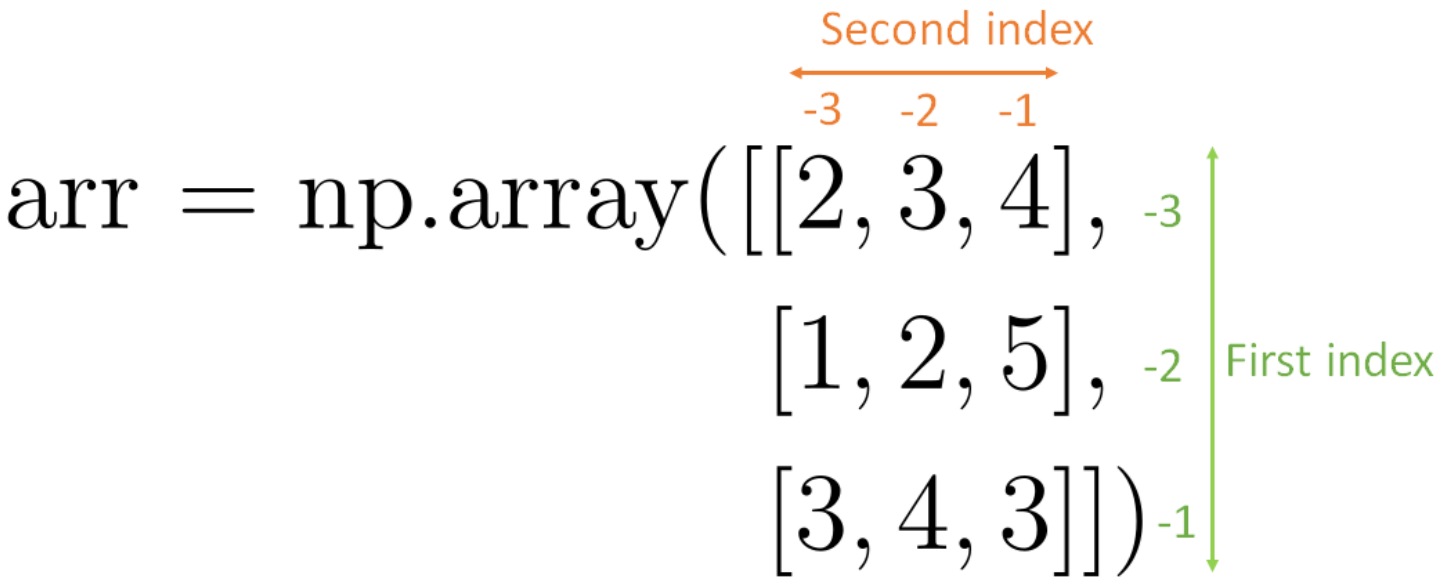
One-dimensional Negative Indexing




```
import numpy as np

arr= np.array([4,7,2])

print(arr)
print("\n")

print(arr[-3])
```

```
[4 7 2]


4
```

Two dimensional Negative Indexing

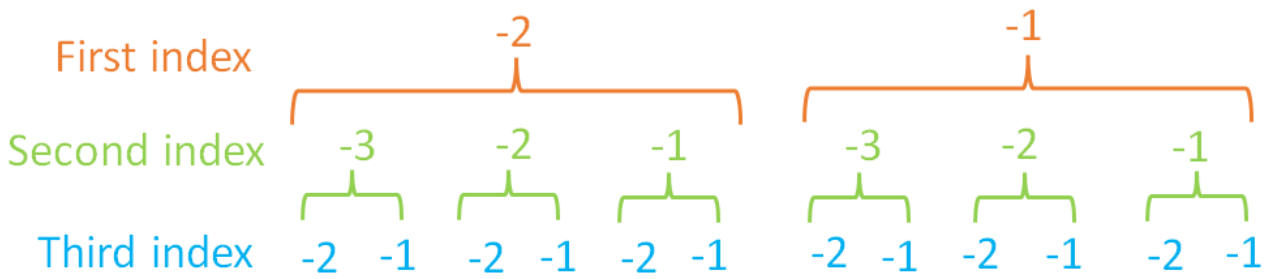



```
import numpy as np

arr= np.array([[2,3,4],[1,2,5],[3,4,3]])
print(arr)
print("\n")

print(arr[-1,-2])
```

```
[[2 3 4]
```

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


4
```

Three-dimensional Negative Indexing



$$\text{arr} = \text{np.array}([[[2,1],[4,5],[6,2]],[[3,4],[7,6],[10,2]]])$$

.

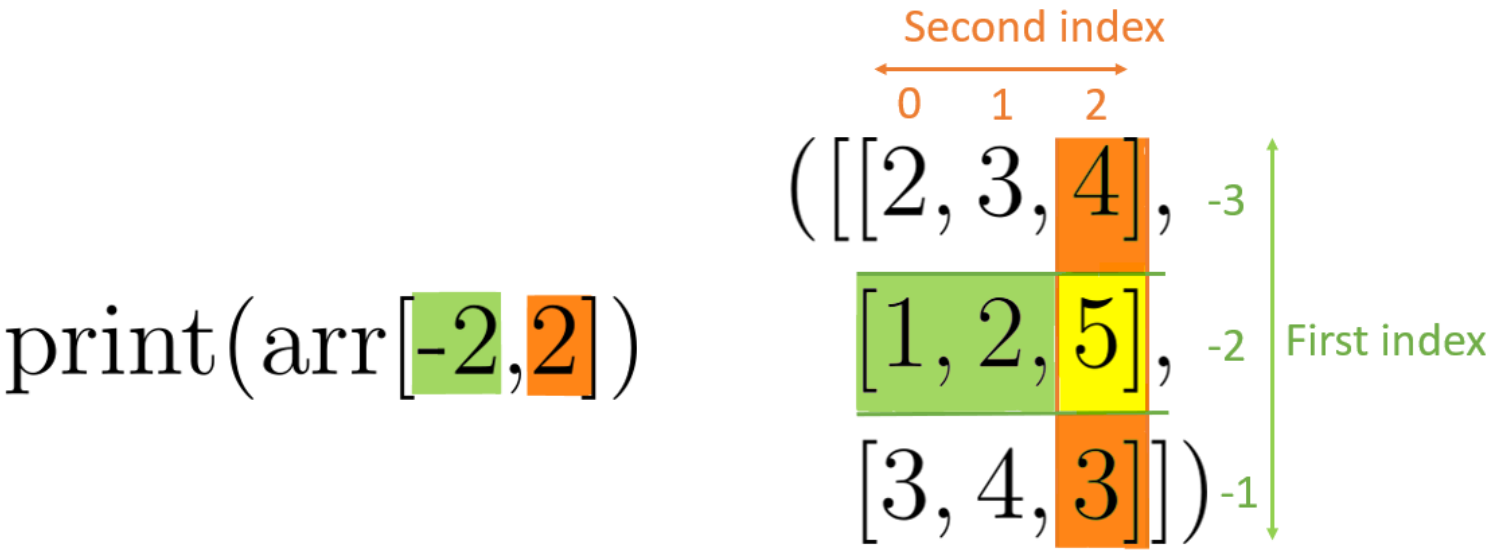
```
import numpy as np

arr= np.array([[[2,1],[4,5],[6,2]],[[3,4],[7,6],[10,2]]])

print(arr)
print("\n")

print(arr[-1,-2,-1])
```

```
[[[ 2  1]
  [ 4  5]
  [ 6  2]]

 [[ 3  4]
  [ 7  6]
  [10  2]]]


6
```

Combining positive and negative indexes

> You can always combine positive and negative indexes to select an element in an array.

> For example if I want to print the number 5 from the following array, I can index it with -2 as first index and 2 as second index.



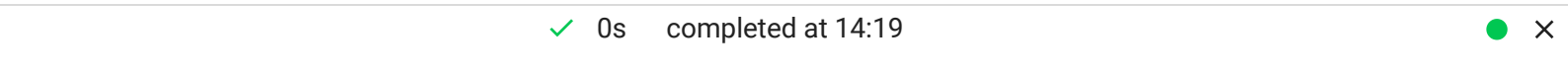

```
import numpy as np

arr= np.array([[2,3,4],[1,2,5],[3,4,3]])
print(arr)
print("\n")

print(arr[-2,2])
```

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


5
```

Calculate the prime numbers between 0 and 100 by using a Boolean array.

```
import numpy as np

is_prime = np.ones((100,), dtype=bool)

# Cross out 0 and 1 which are not primes:
is_prime[:2] = 0

# cross out its higher multiples (sieve of Eratosthenes):
nmax = int(np.sqrt(len(is_prime)))
for i in range(2, nmax):
    is_prime[2*i::i] = False

print(np.nonzero(is_prime))
```

```
(array([ 2,  3,  5,  7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59,
       61, 67, 71, 73, 79, 83, 89, 97]),)
```


0s
completed at 14:19