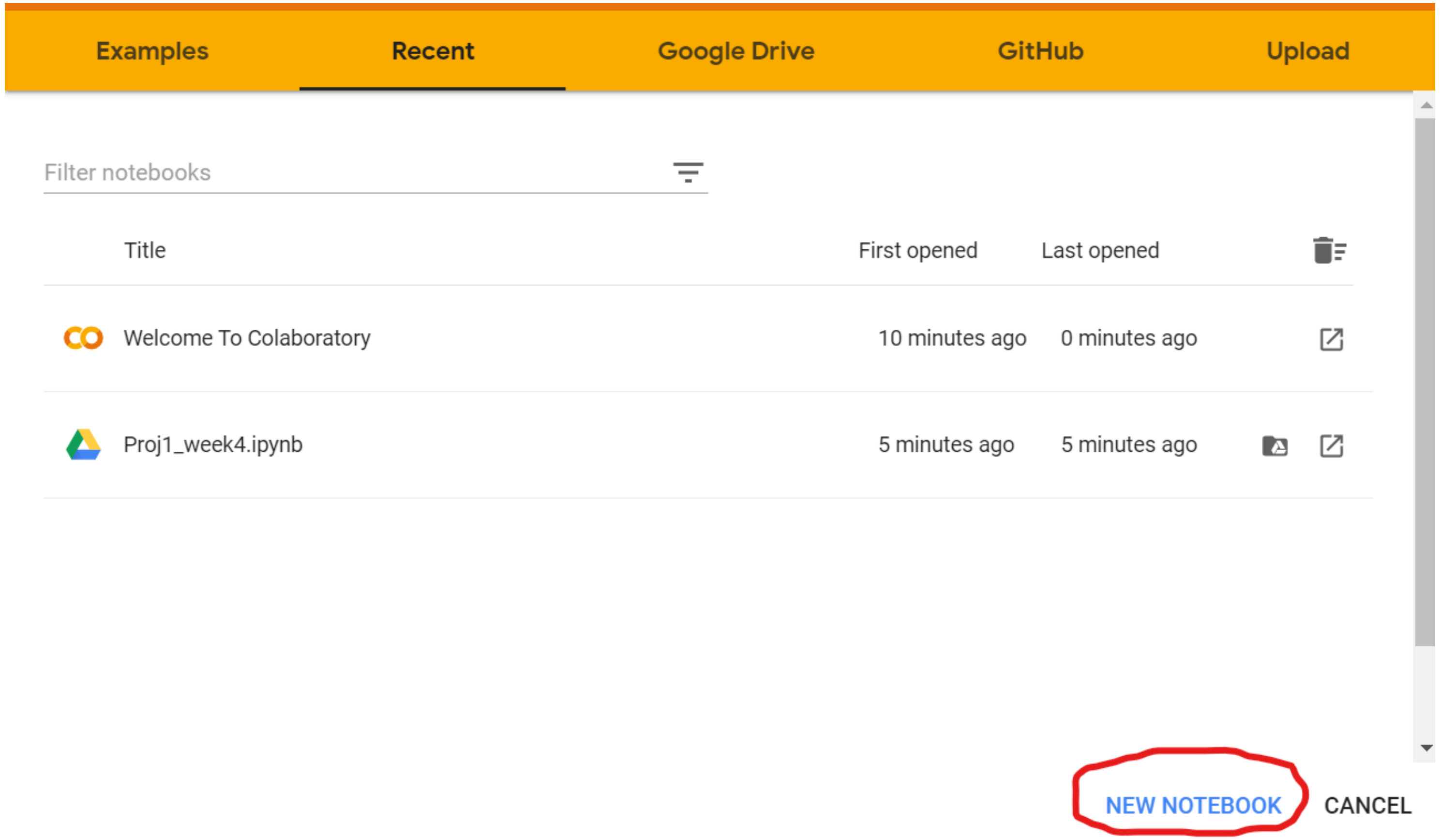
PRIME INTUIT
Finishing School

# Getting started with Google Colab

# Getting started with Google Colab

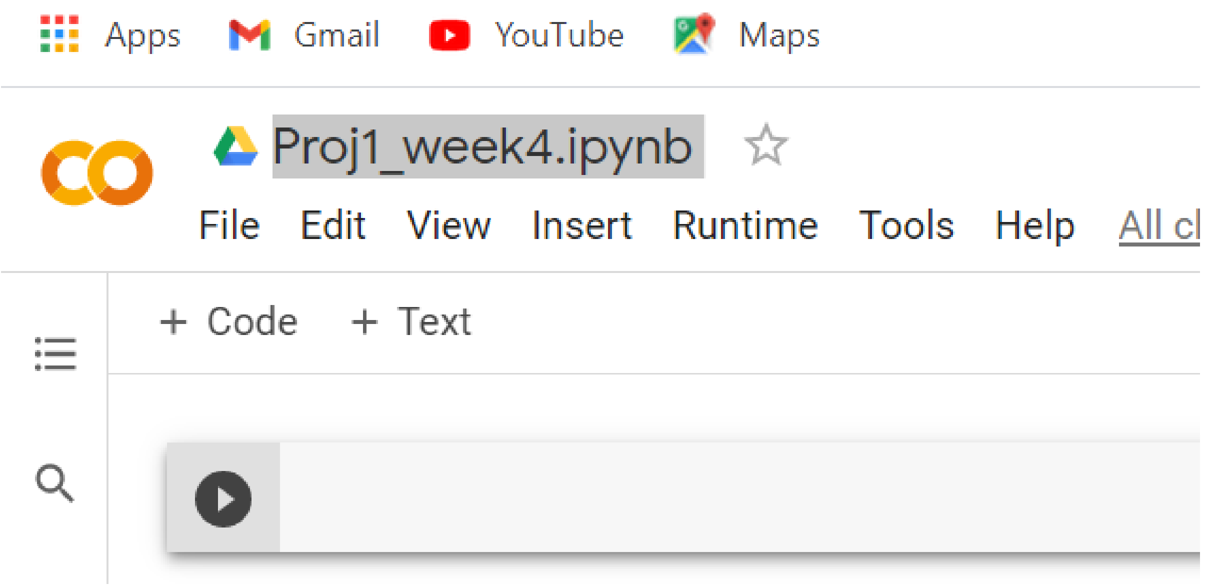
**https://colab.research.google.com**



PRIME INTUIT
Finishing School

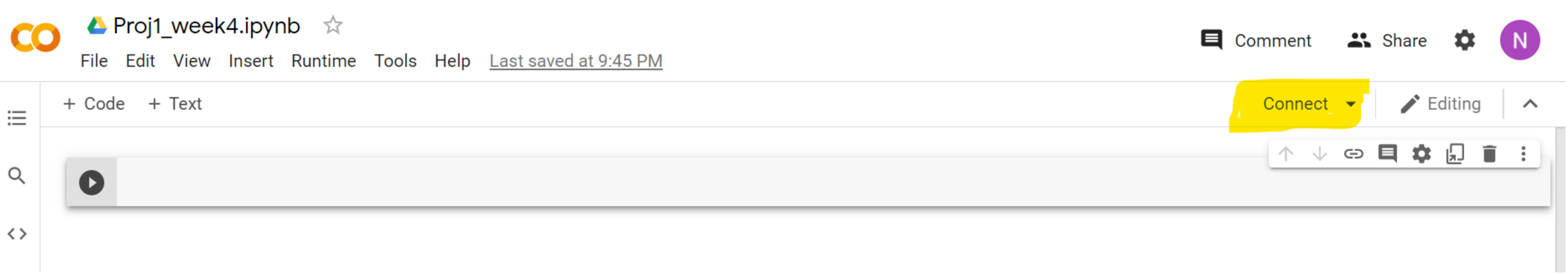
# Getting started with Google Colab

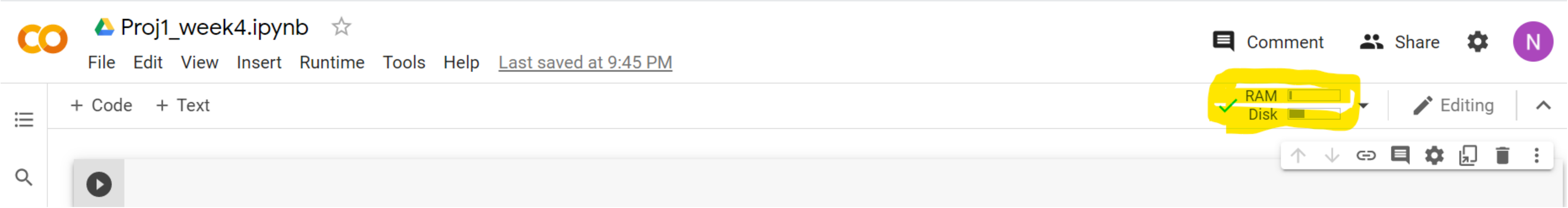
**https://colab.research.google.com**



# Getting started with Google Colab

https://colab.research.google.com



PRIME INTUIT
Finishing School

# Numpy

PRIME INTUIT
Finishing School

# NumPy

## Working with arrays

- ✓ Scientific Computing
- ✓ Financial analysis
- ✓ Relational data
- ✓ Multimedia
- ✓ Machine learning

**All these require storing and processing of high dimensional arrays efficiently**

# NumPy

## We have already seen lists, tuples, sets , dictionaries

**Lists can store collection of high dimensional arrays and we can operate on them by iterations**

# NumPy

## We have already seen lists, tuples, sets , dictionaries

**Lists can store collection of high dimensional arrays and we can operate on them by iterations**

**But this is very inefficient and slower then expected performance: 10X to 100X**

**Why ?**
**Lists are designed to store heterogeneous data**
**No low level hardware mechanisms to accelerate the operations on lists**

# NumPy

NumPy

**Intended to bring performance and functionality improvements for numerical computing**

**Started in 2006**

**Now a standard package used in many real world applications, other packages**

POWERFUL N-DIMENSIONAL ARRAYS
Fast and versatile, the NumPy vectorization, indexing, and broadcasting concepts are the de-facto standards of array computing today.

NUMERICAL COMPUTING TOOLS
NumPy offers comprehensive mathematical functions, random number generators, linear algebra routines, Fourier transforms, and more.

# NumPy

Intended to bring performance and functionality improvements for numerical computing

- ✓ Efficiently stores n-d arrays in vectorized form to benefit from DRAM locality
- ✓ Enables easy file save and load of n-d arrays
- ✓ Efficiently process data without type checking overhead
- ✓ Enable other packages to use numpy arrays as an efficient data interface
- ✓ Efficiently broadcast operations across dimensions
- ✓ Provide implementations of many functions across linear algebra, statistics, etc.

PRIME INTUIT
Finishing School

# NumPy

NumPy

## What are we interested in:

- ✓ What are n-d arrays
- ✓ What is broadcasting
- ✓ How to load and save n-d arrays
- ✓ How to use statistical functions

# NumPy

# Comparing performance with lists....

```
N = 1000000000

%%time
List1 = list(range(N))
For I in range(N)
List1[i] = List1[i] * List1[i]

%%time
List1 = list(range(N))
List1 = [item * item for item in List1]
```

PRIME INTUIT
Finishing School

# NumPy

```
%%time
List1 = list(range(N))
List1 = map(lambda x: x * x, List1)

%%time
List1 = list(range(N))
List_sum = 0
For item in List1
List_Sum + =  item

%%time
List1 = list(range(N))
List1_sum = sum(
```

PRIME INTUIT
Finishing School

# NumPy

```
Imprt numpy as np
%%time
Arr =np.arrange(N)
Arr = Arr * Arr

%%time
Arr =np.arrange(N)
Arr_sum = np.sum(Arr)
```

# High Dimensional Array & Creating NumPy Array

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | 4 |

One Dimension Array

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

Two Dimension Array

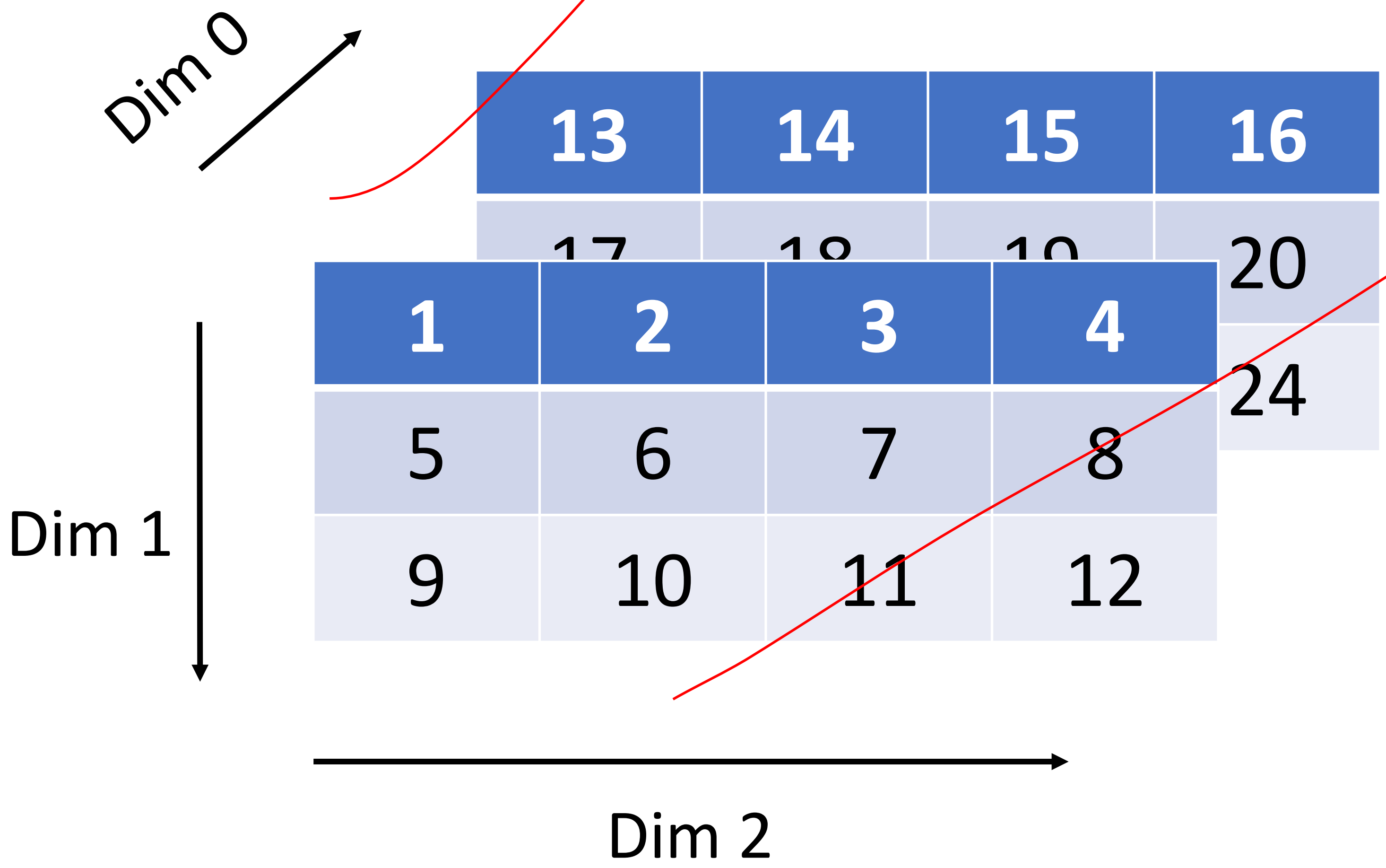
| 1 | 2 | 3 | 4 |
|---|---|---|---|
| 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 |

| 13 | 14 | 15 | 16 |
|---|---|---|---|
| 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 |

Three Dimension Array

PRIME INTUIT
Finishing School

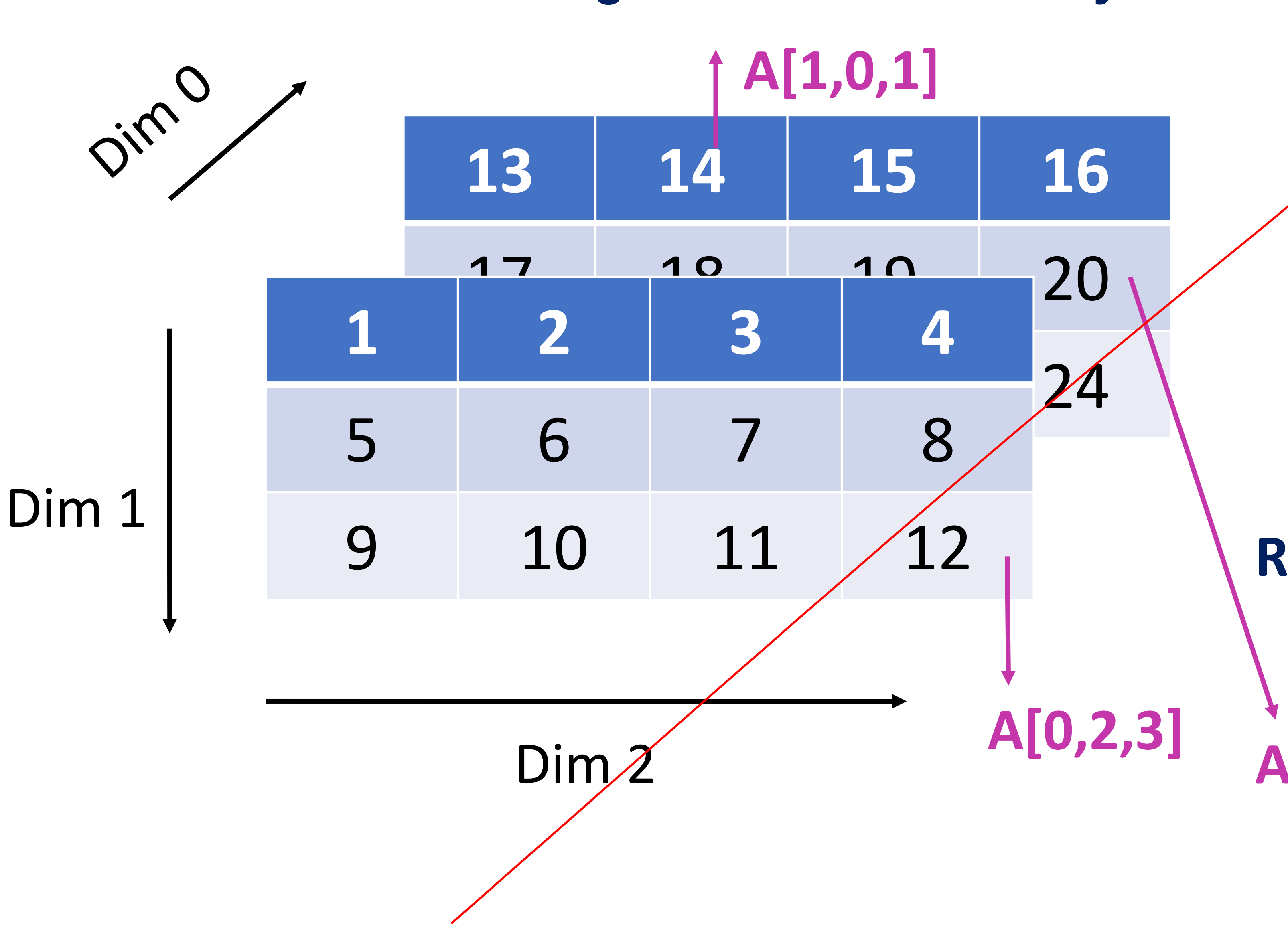
# High Dimensional Array & Creating NumPy Array




**We index dimensions backwards in order we added them**

**So we have a 3 Dim array of size 2 x 3 x 4 (Shape of array)**

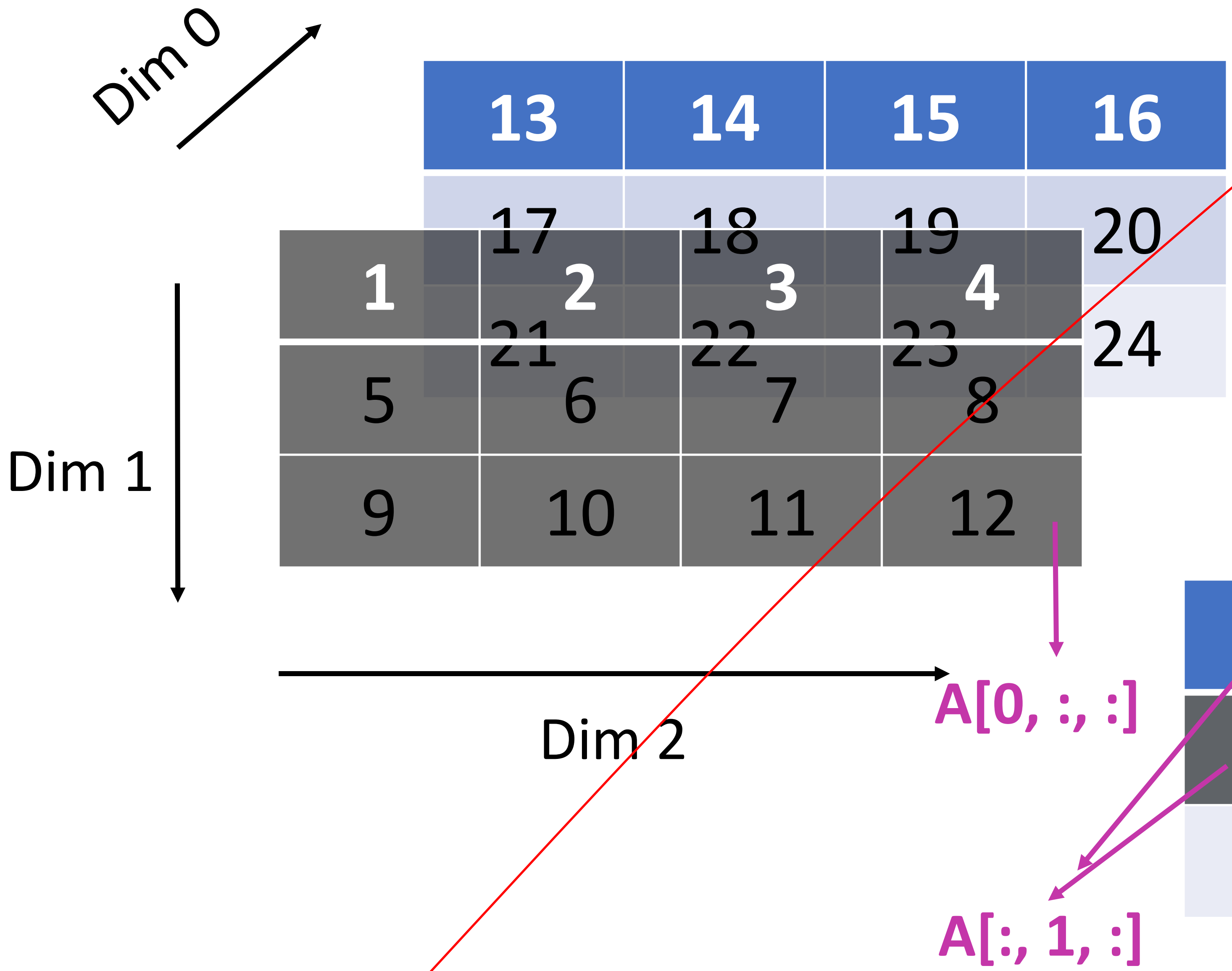
# High Dimensional Array & Creating NumPy Array



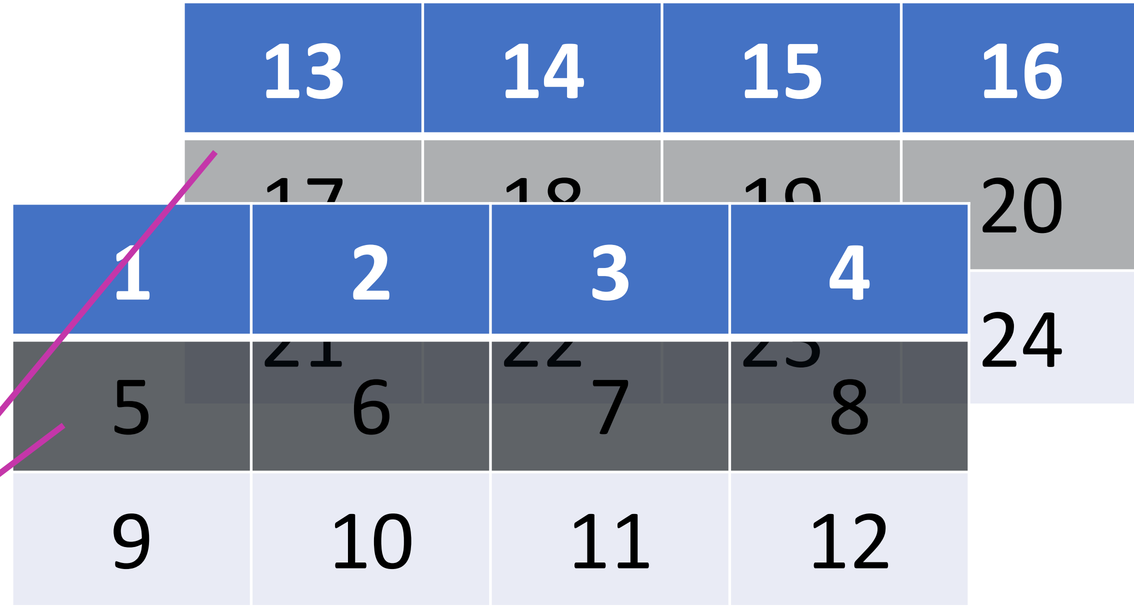

**We can index each item of the 3Dim array by specifying 3 co-ordinates**

**Remember we index from 0**

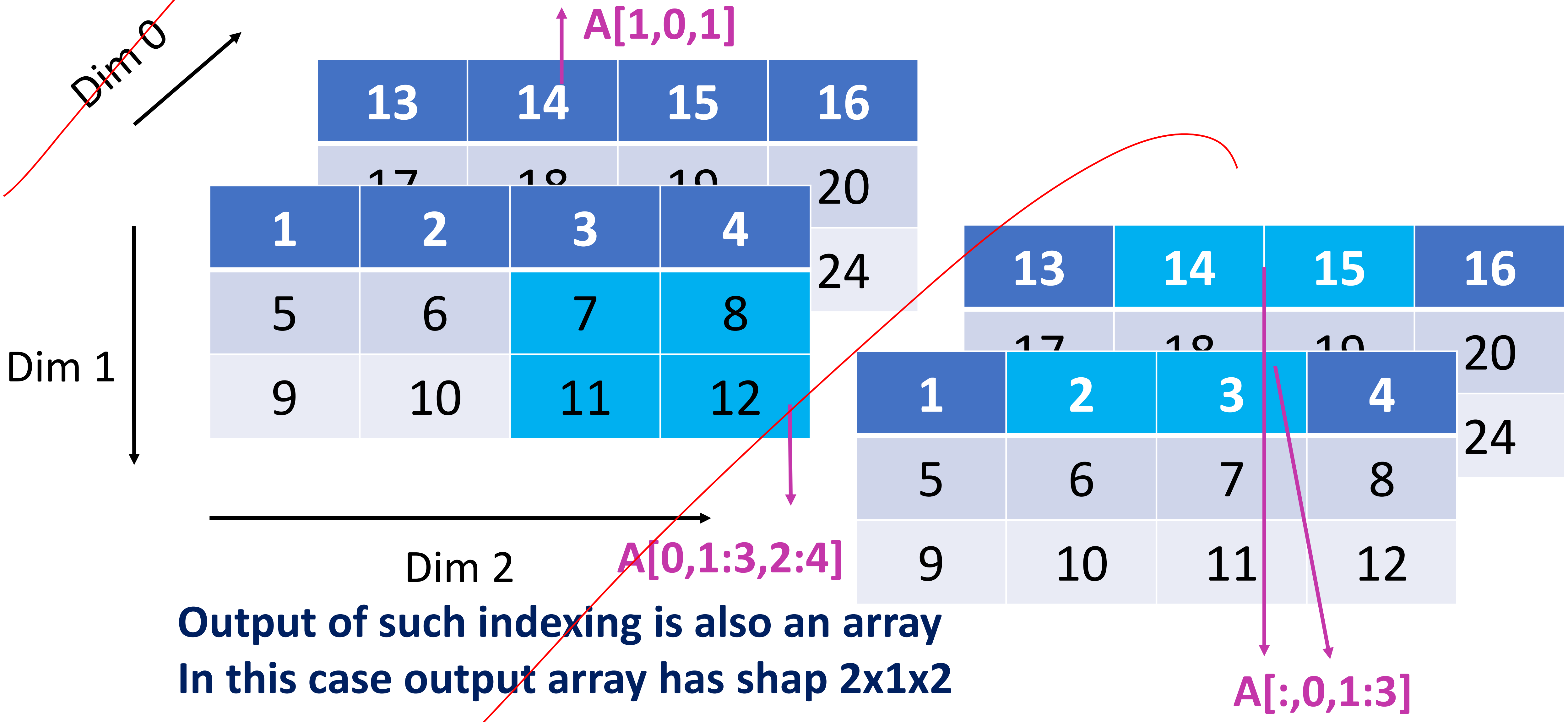
# High Dimensional Array & Creating NumPy Array




**We can refer to slices of the data with partial indices**



## High Dimensional Array & Creating NumPy Array




**Output of such indexing is also an array**
**In this case output array has shap 2x1x2**

# High Dimensional Array & Creating NumPy Array

```
Arr =np.arrange(5)
Print(Arr, type(Arr))


Arr = np.Arr([0, 2, 4, 6, 8])
Print(Arr, type(Arr))

Arr  # this would print an array

Array([0, 2, 4, 6, 8])
Print(Arr, type(Arr))
```

# High Dimensional Array & Creating NumPy Array

```
Arr =np.arrange(5)
Print(Arr, type(Arr))

Arr = np.Arr([0, 2, 4, 6, 8])
Print(Arr, type(Arr))

Arr  # this would print an array
Arr.dtype

Arr.ndim
Arr.shape
Arr.size
```

## High Dimensional Array & Creating NumPy Array

```
Arr.itemsize

#2 dimensional Array

Arr2d = np.array([
                  [1, 2, 3],
                  [4, 5, 6]
])

Arr2d

Arr2d.ndim
```

# High Dimensional Array & Creating NumPy Array

```
Arr2d.shape
Arr2d.size

# 3 dimension array
Arr3d = np.array([
        [
        [1, 2, 3],
        [4, 5, 6]
        ],
        [
        [7, 8, 9],
        [10, 11, 12]
        ]
])
```

# High Dimensional Array & Creating NumPy Array

```
Arr3d.shape
Arr3d.ndim
Arr3d.size

# Other arrays that can be created

np.ones((3, 4))
Np.zeros((2, 3, 4))

2010 * np.ones((2,3,2))
```

# High Dimensional Array & Creating NumPy Array

**# Random Array's**

**np.random.randn(2, 3)**

**Np.random.rand(2, 3)**

**np.random.randint(0, 100 (2, 3)**

**np.arrange(7, 71, 7)**

**np.linspace (7, 70, 10)**

# High Dimensional Array & Creating NumPy Array

# Array's of other kind

```
np.array([True, False, True, False])

np.array([ '1.4',  '1.6',  '1.8'])


Str_arr = np.array([ '1.4',  '1.6',  '1.8'])
Arr1 = np.array(str_arr, dtype = 'float')

Arr1
```

PRIME INTUIT
Finishing School

# Indexing array

```
# Indexing of Array's
Arr3d = np.array([
                    [
                    [1, 2, 3],
                    [4, 5, 6]
                    ],
                    [
                    [7, 8, 9],
                    [10, 11, 12]
                    ]
])
Print(Arr3d)

Arr3d[0, 0, 0]
Arr3d[1, 0, 2]
```

# Indexing array

```
# Indexing of Array's
Arr3d = np.array([
                    [
                    [1, 2, 3],
                    [4, 5, 6]
                    ],
                    [
                    [7, 8, 9],
                    [10, 11, 12]
                    ]
])
Print(Arr3d)

Arr3d[0, 0, 0]
Arr3d[1, 0, 2]
```

# Indexing array

**# Indexing of Array's**

**I = 1**
**J = 2**
**K = 0**

**Arr3d[I, j, k]**

**Arr3d[0, :, :]**

**Arr3d[1, :, :]**

**Arr3d[:, 1, :]**

**Arr3d[;, ;, 0:2]**

# Indexing array

```
# Fancy indexing

Arr3d % 2 == 0

Arr3d[Arr3d % 2 == 0 ]

Arr3d[Arr3d % 2 == 1 ]

Arr3d[(Arr3d % 2 == 1) & (Arr3d > 3)]

Arr_Slice = Arr3d[:, :, 0:2]

Print(type(Arr_Slice))
```

# Indexing array

Arr_Slice.ndim

Arr_Slice.shape

Arr_Slice[0, 0, 1]

Arr_Slice[0, 0, 1] = 1999
Arr_Slice

Arr3d # original array is also updated as its not a deep copy but only ref to the number is changed.

How to tackle this

Arr_Slice = np.copy(Arr3d[:, :, 0:2])

# Indexing array

**How to tackle this**

**Arr_Slice = np.copy(Arr3d[:, :, 0:2])**

**Arr_Slice[0, 0, 1] = 1**

**Arr_Slice**
**Arr3d # Remains the same as we did a deep copy**

**Arr = np.random.randint(0, 10, (5))**
**Arr**
**My_List[1, 3, 4]**
**Arr[My_List]**

# Numpy Operations

```
Arr1 = np.zeros((3,4))
Arr2 = np.ones((3,4))

Arr1 + Arr2

Arr3 = np.random.rand(3,4)
Arr4 = np.random.rand(3,4)
Arr3
Arr4
Arr3 + Arr4

Arr3 – Arr4
Arr3 * Arr4
Arr3 / Arr4
```

PRIME INTUIT
Finishing School

# Numpy Operations

# Operations on a single Array

Np.exp(Arr3)   # exponent the values in array e**x , X is the element in particular location of the array

Np.log(Arr3)    # returns the log of the array element

Np.log(np.exp(Arr3))  # check log and exponent

Np.sin(Arr1)

Np.cos(Arr2)

Np.sqrt(Arr3)

# Numpy Operations

```
# Operations on a single Array

Arr_inv = 1 / Arr3

Arr4 = np.zeros((3,4))
Arr_inv1 = 1 / Arr4

Print(Arr_inv1)

# inf referred to infinity

Np.isinf(Arr_inv1[0,0])

Np.isinf(arr_inv)
```

# Numpy Operations

Get the common items between two numpy arrays

a = np.array([1,2,3,2,3,4,3,4,5,6])
b = np.array([7,2,10,2,7,4,9,4,9,8])

array([2, 4])

```
a = np.array([1,2,3,2,3,4,3,4,5,6])
b = np.array([7,2,10,2,7,4,9,4,9,8])
np.intersect1d(a,b)
```

# Numpy Operations (square and circle exercise)

```
Import numpy as np
Ndim = 2
Npoints = 100000
Points = np.random.rand(npoints,ndim)

dfo = np.zeros((npoints, 1))
Outside_points = 0
For I in range(npoints)
        for j in range(ndim)
                dfo[i] += point[I,j]**2
                dfo[i] =np.sqrt(dfo[i])
        If dfo[i] > 1:
        Outside_points += 1

Print('Fraction of points outside is ', outside_points/npoints)
```

PRIME INTUIT
Finishing School

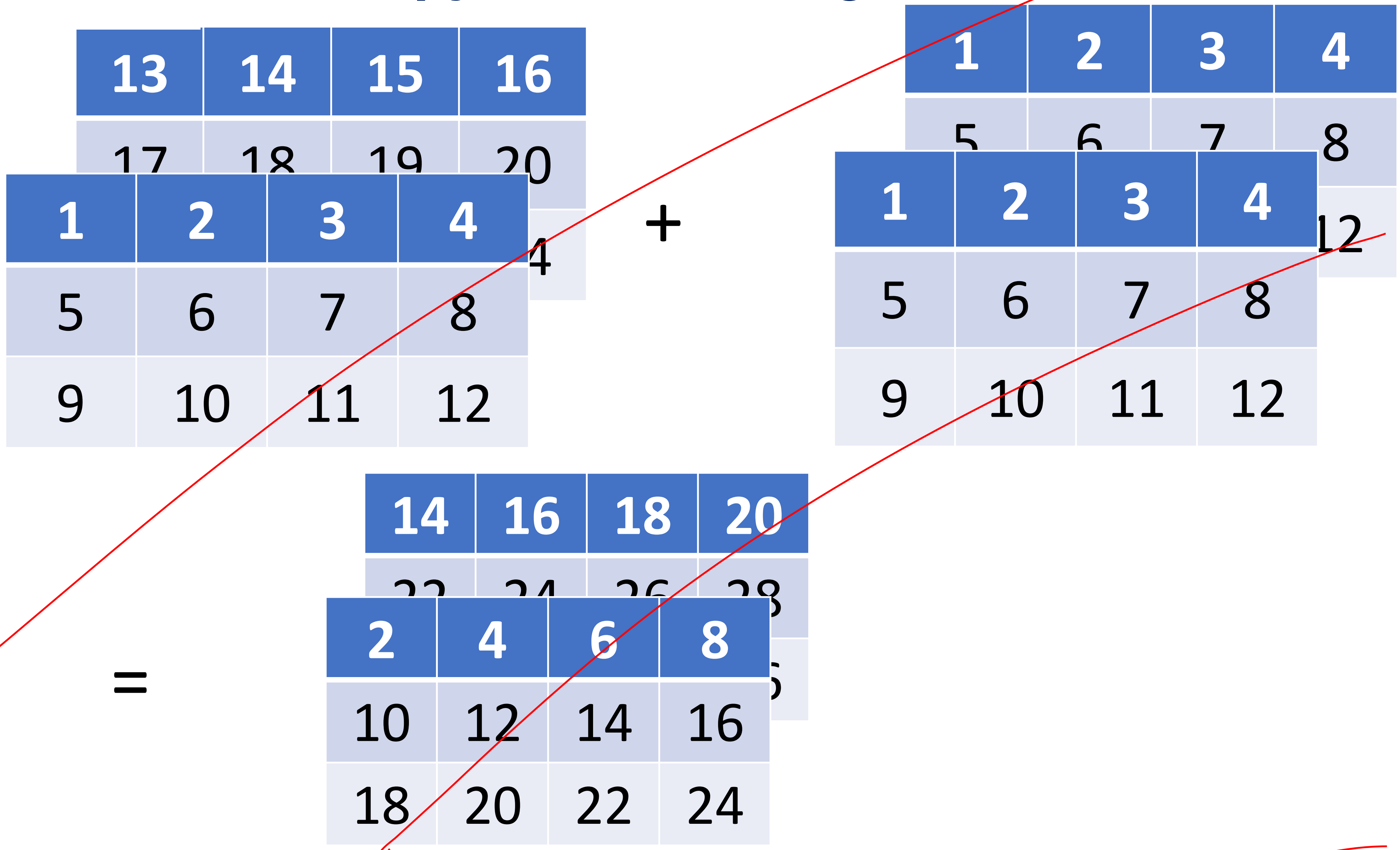
# Numpy Operations (square and circle exercise)

```
Import numpy as np
Ndim = 2
Npoints = 100000
Points = np.random.rand(npoints,ndim)

dfo = np.zeros((npoints, 1))
Outside_points = 0
For I in range(npoints)
        for j in range(ndim)
                dfo[i] += point[I,j]**2
                dfo[i] =np.sqrt(dfo[i])
        If dfo[i] > 1:
        Outside_points += 1

Print('Fraction of points outside is ', outside_points/npoints)
```

# Numpy Broadcasting



# Numpy Broadcasting

| | | | |
|---|---|---|---|
| 13 | 14 | 15 | 16 |
| 17 | 18 | 19 | 20 |
| | | | 4 |

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

+

| | | | |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 |
| | | | 2 |

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

=

| | | | |
|---|---|---|---|
| 14 | 16 | 18 | 20 |
| 22 | 24 | 26 | 28 |
| | | | 6 |

| | | | |
|---|---|---|---|
| 2 | 4 | 6 | 8 |
| 10 | 12 | 14 | 16 |
| 18 | 20 | 22 | 24 |

PRIME INTUIT
Finishing School

# Numpy Broadcasting

| | | | |
|---|---|---|---|
| 13 | 14 | 15 | 16 |
| 17 | 18 | 19 | 20 |
| | | | 4 |

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

+

| | | | |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 5 | 5 | 5 | 5 |
| | | | 9 |

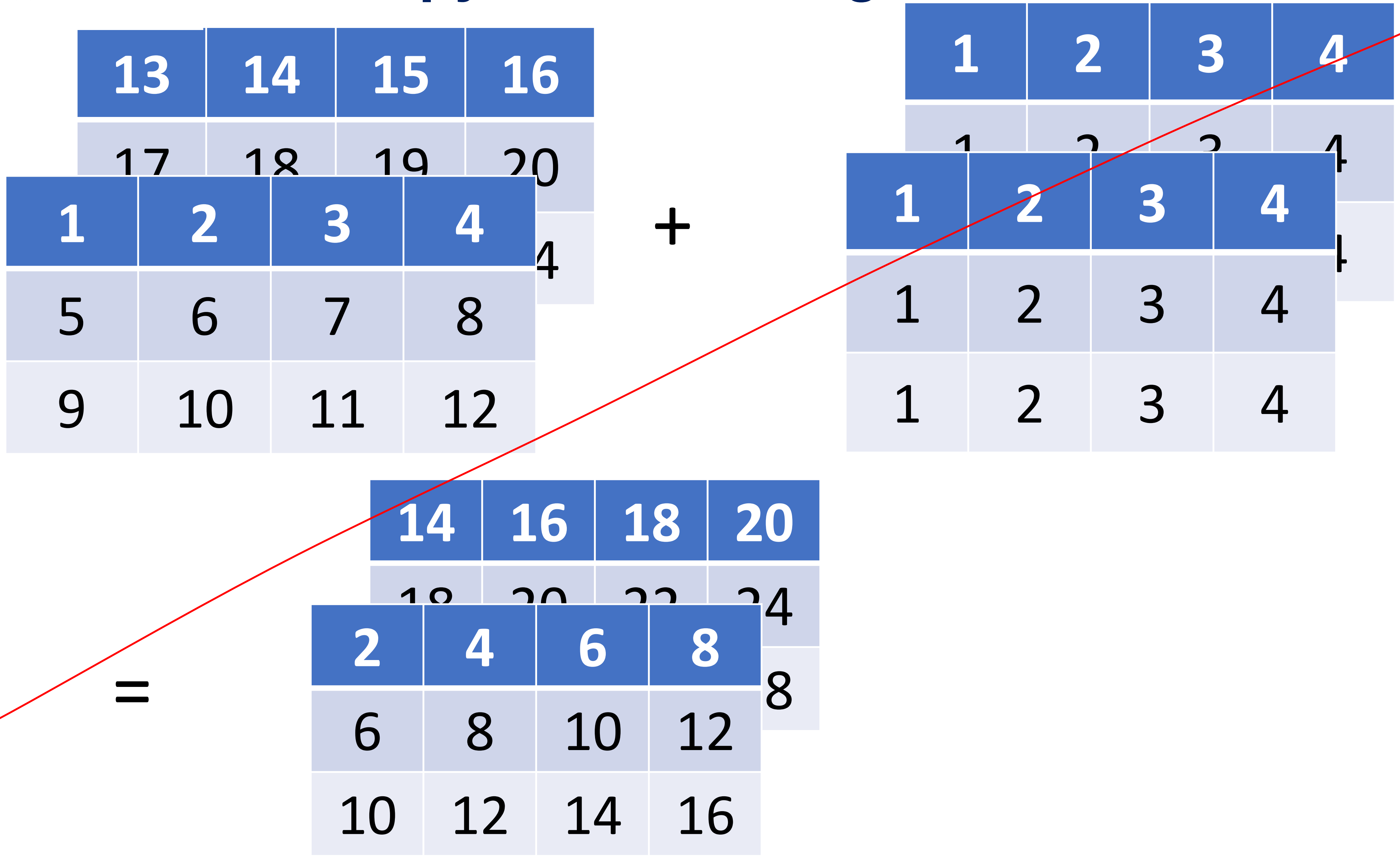
| | | | |
|---|---|---|---|
| 1 | 2 | 3 | 4 |
| 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 |

=

| | | | |
|---|---|---|---|
| 14 | 15 | 16 | 17 |
| 22 | 23 | 24 | 25 |
| | | | 3 |

| | | | |
|---|---|---|---|
| 2 | 3 | 4 | 5 |
| 10 | 11 | 12 | 13 |
| 18 | 19 | 20 | 21 |

# Numpy Broadcasting



Finishing School

# Numpy Broadcasting

| | | | |
|---|---|---|---|
| 13 | 14 | 15 | 16 |
| 17 | 18 | 19 | 20 |
| | | | 24 |

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

+

| | | | |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |
| | | | 1 |

| | | | |
|---|---|---|---|
| 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |

=

| | | | |
|---|---|---|---|
| 14 | 15 | 16 | 17 |
| 18 | 19 | 20 | 21 |
| | | | 25 |

| | | | |
|---|---|---|---|
| 2 | 3 | 4 | 5 |
| 6 | 7 | 8 | 9 |
| 10 | 11 | 12 | 13 |

PRIME INTUIT
Finishing School

# Numpy Broadcasting

| 1 | 2 | 3 | 4 |
|---|---|---|---|

+

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|

| 1 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|
| 2 | 2 | 2 | 2 | 2 |
| 3 | 3 | 3 | 3 | |
| 4 | 4 | 4 | 4 | |

+

| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 |
| 1 | 2 | 3 | 4 | 5 |
| 1 | 2 | 3 | 4 | 5 |

=

| 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|
| 3 | 4 | 5 | 6 | 7 |
| 4 | 5 | 6 | 7 | 8 |
| 5 | 6 | 7 | 8 | 9 |

PRIME INTUIT
Finishing School

# Numpy Broadcasting ( Reshape)

```
Import numpy as np

Arr_1 =np.arrange(6)
Arr_1.shape
Arr_1
Arr_1 = Arr1.reshape((3,2))
Arr_1.shape
Arr_1

Arr_2 =np.arange(6).reshape((3,2))
Arr_2

Arr1 + Arr_2
```

(3,2)
(3,2)

# Numpy Broadcasting

```
Arr_2[0].reshape((1,2))

Arr_1 + Arr_2[0].reshape((1,2))      ## (3,2) + (1,2) broadcasting

Arr_2[:, 0].reshape((3,1))

Arr_1 + Arr_2[:, 0].reshape((3,1))      ## (3,2) + (3,1) broadcasting

Arr_1 + 1                          ## (3,2) + (1) scalar broadcasting

Arr_3 = np.arange(24).reshape((2, 3, 4))

Arr_3
```

# Numpy Broadcasting

```
Arr_4 = np.ones((1,4))
Arr_4
```

2,3,4
1,4

```
Arr_3 + Arr_4   ## (2, 3, 4) + (1, 4)
```

```
Arr_5 = np.arange(4)
Arr_5
Arr_6 = np.arange(5)
Arr_6
```

error

0,0,4
0,0,5

```
Print(Arr_5.shape, arr_6.shape)
Arr_5 + Arr_6      ### in compatable and needs tranpose
```

```
Arr_5.reshape(4, 1) + Arr_6  ##(4,1) + (5)
```

PRIME INTUIT
Finishing School

# Numpy File Handling

**Files / upload /Planets**

```
Planets_Small = np.loadtxt("Planets_Small.txt")

## could not convert string to float

Planets_Small = np.loadtxt("Planets_Small.txt", skiprows = 1)

Planets_Small = np.loadtxt("Planets_Small.txt", skiprows = 1,
                                              usecols =(1, 2, 3, 4, 5, 6, 7, 8, 9))

Planets_Small
Planets_Small.ndim
Planets_Small.shape
```

# Numpy File Handling

Files / upload /Planets

```
Planets = np.loadtxt("Planets.txt")

## could not convert string to float unknown

Planets = np.genfromtxt("Planets.txt", skip_header = 1,
                        usecols = [1, 2, 3, 4, 5, 6, 7, 8, 9])
Planets
Planets.shape
Planets.ndim

Np.isnan(planets)
Planets_New = np.nan_to_num(planets, nan = 1)
Planets_New
```

# Numpy File Handling

```
Np.savetxt('Planets_New.txt', Planets_new, delimiter = ',')

Np.save('Planets_new', Planets_new)

!ls

!ls –lh

Arr_a = np.random.rand(1000, 2)
Arr-b = np.random.rand(2000, 5)
Arr_c = np.random.rand(20, 10000)

Np.savez("manyarrays", arr_a, arr_b, arr_c)

!ls -1
```

# Numpy File Handling

```
Arrs = np.load('manyarrays.npz')
Print(type(arrs))

Arrs.files

Arrs['arr_a']

Arrs['arr_a'].shape
Np.savez_compressed('many_arr_comp', arr1, arr2, arr3)

Arrs_d = np.zeros((10000, 10000))
Np.savez("Zeros", arr_d)
Np.savez_comp("Zeros_compressed", arr_d)

!ls -lh
```

# Numpy Exercises

**How to stack two arrays horizontally?**

```
a = np.arange(10).reshape(2,-1)

b = np.repeat(1, 10).reshape(2,-1)

np.concatenate([a, b], axis=1)
```

Or

```
np.hstack([a, b])
```

# Stats with Numpy

# Stats with Numpy

```
import numpy as np
Create a large array to work with
a1 = np.random.rand(100000,)

np.min(a1)
np.max(a1)
np.mean(a1)
np.var(a1)
np.std(a1)
np.median(a1)
np.percentile(a1, 50)
```

# Stats with Numpy

```
np.percentile(a1, 25)
iqr= np.percentile(a1,75) - np.percentile(a1,25)
print(iqr)
quartile = np.percentile(a1, [25, 75])
print(quartile)
iqr = quartile[1] - quartile[0]
print(iqr)
# use %% time to compute

# Z Score
zscore = (a1 -np.mean(a1))/np.std(a1)
print(zscore)
print(zscore.mean())
```

# Stats with Numpy

```
np.histogram(a1)
np.histogram(a1, bins=5)
np.histogram(a1, bins = [.20, .4, .6, .8])

Mapping points to bins, which point in my array lies
in which bin
Bins = [0, 0.25, .5, .75, 1]
Np.digitize(a1,bins)

Left boundary inclusion
A2 = np.random.randint(0,10,(10))
a2
Bins = [0,6,10]
Np.digitize(a2,bins)
Np.digitize(a2,bins, right = true)
```

# Stats with Numpy

```
a3 = np.random.randint(40, 90, 100)
a4 = np.random.randint(150, 185, 100)
a5 = np.random.randint(17, 30, 100)
Np.concatenate((a3, a4, a5))
Np.concatenate((a3, a4, a5)).shape
np.vstack((a3, a4, a5))
np.vstack((a3, a4, a5)).shape
np.hstack((a3, a4, a5))
np.hstack((a3, a4, a5)).shape
a6 = np.vstack((a3, a4, a5))
np.amin(a6, axis = 1)
np.amax(a6, axis = 1)
np.mean(a6, axis = 1)
```

# Rules of Statistics

```
# checking rules of statistics with Numpy
# mean subtracted array has zero mean

a7 = np.random.rand(1000)
mean = np.mean(a7)
a8 = a7 - mean
np.mean(a8)

a9 = np.random.rand(1000)
for k in range (1,50):
  a10 = a9[0:k]
  print(k, np.mean(a10))
```

# Rules of Statistics

```
#Alternative way and finding help
np.cumsum?
np.cumsum(a9)/ np.arange(1,1001)

# Effect of outliers on mean and median

a11 = np.random.randint(1, 100, 100)
np.mean(a11)
np.median(a11)

a12 = np.append(a11, [1000, 2000])
a12.shape
np.mean(a11)     # sensitive to outliers
np.median(a11)   # not so sensitive to outliers
```

# Rules of Statistics

```
# effect of scaling on mean and median
a13 = np.random.rand(100)
np.mean(a13)
np.median(a13)
# x = xa + C
a14 = 2.5 * a13 + 0.89
print(np.mean(a14), (2.5 * np.mean(a13)+0.89))
print(np.median(a14), (2.5 * np.median(a13)+0.89))

print(np.var(a14), (2.5 *2.5* np.var(a13)))
print(np.std(a14), (2.5 *2.5* np.std(a13)))
```

PRIME INTUIT
Finishing School

# Numpy Mini Project

**Cric_data**
**!head cric_data.tsv**

1) Injust the data into array, find mean, median, IQR for Sachin, Rahul and India
2) Find the histogram of sachin's Scores with 10 bins
3) Find mean of sachin's scores grouped by 25 matches
4) Find mean of sachin's scores where he has scored a century
5) Find mean of sachin's scores when Rahul has scored less then 10
6) Find mean of sachin's scores based on which quartile India's score falls in
7) For every match findout who has scored more Sachin or Rahul
8) How many more runs does sachin score on an average after scoring X runs
9) How many matches did sachin take to score first 1000 runs and then next 1000 …..

# Numpy Mini Project

```
cric_data = np.loadtxt("cric_data.tsv", skiprows = 1)

Cric_data.shape
Cric_data = cric_data[:,[1,2,3]]

Sachin = cric_data[:, 0]
Rahul = cric_data[:, 1]
India = cric_data[:, 2]

# problem 1
Def stats(col):
        print(“mean”, np.mean(col))
        print(“median”, np.median(col))
        print(“IQR”, np.percentile(col,75) –np.percentile(col,25))

Stats(sachin)
```

# Numpy Mini Project

```
Stats(Rahul)
Stats(India)

#alternatively
Np.mean(cric_data, axis = 0)
Np.median(cric_data, axis = 0)
Np.percential(cric_data, 75,axis =0) – np.percentail(cric_data, 25, axis =0)

# problem 2

Np.histogram(Sachin)
```

PRIME INTUIT
Finishing School

# Numpy Mini Project

```
# Problem 3

Sachin.shape
Sachin.reshape(0,25), shape
Sachin_25 = Sachin.reshape(0,25), shape

Np.mean(sachin_25, axis = 1)

Problem 4 & 5

Sachin > = 100
sachin[sachin > = 100])
Np.mean(sachin[sachin > = 100])

Np.mean (sachin[Rahul<=10)
```

PRIME INTUIT
Finishing School

# Numpy Mini Project

```
# problem 6
Np.percentile(india[25,50,75,100])
Qrt = Np.percentile(india[25,50,75,100])

India<175

India < = qrt # this wont work, so we seek help of broadcasting
India.shape
Qrt.shape
Qrt = qrt.reshape(4,1)
India < qrt
Indices = india<qrt
Indices.shape
Sachine[indices[0,:1]
```

PRIME INTUIT
Finishing School

# Numpy Mini Project

```
# problem 6

Sachine[indices[1,:1]

For I in range(4)
Print(I, np.mean(sachin[indices[i]]))

# problem 7
Snr = cric_data[:, 0:2]
Np.max([1,3,2,5,1])
Np.argmax([1,3,2,5,1])
Np.max([10,3,2,5,1])
Np.argmax([10,3,2,5,1])
```

# Numpy Mini Project

# problem 7

Is_Rahul_higher =Np.argmax(snr, axis =1)
Np.sum(is_Rahul_higher) /225

# np.where func
Np.where(is_Rahul_higher ==0, 'Sachin', 'Rahul')

# Numpy Mini Project

```
# problem 8

X_arr = np.arange(0,101,5)
Sachine >= x_arr

# reshape and take adv of broadcast
X_arr=x_arr.reshape(x_arr.shape[0],1)

Indices = Sachine >=x_arr

Indices.shape
Sachin[indices[1,:]]

For in in range(x_arr.shape[0]]:
Print(x_arr[I,0], np.mean(sachin[indices[I,:]]) –x_arr[I,0]
```

# Numpy Mini Project

# problem 9

Sachin_cumsrc =Np.cumsum(sachin)

Np.histogram(sachin_cumsrc, bins = np.arrage(0,10000, 1000)

# Numpy Recap

- Understanding Numpy
- Compare performance of Numpy array's to list
- Creating Numpy arrays
- Indexing arrays
- Numpy Operations
- Exercises
- Broadcasting
- Statistics with Numpy
- Checking Statistics Rules with Numpy
- How to apply Numpy on real world problems

PRIME INTUIT
Finishing School

# Numpy Recap

- Data science in several verticals like scientific computing, Financial analysis, relational data, multimedia data and deploring requires storing and processing high dimensional arrays efficiently
- In python we do this using lists, sets, tuples, dictionaries and numpy arrays. For performance, Numpy arrays were significantly faster.
- Provides efficient low – level storage and operations on multi-dim typed arrays
- Provides many efficient indexing methods
- Provides efficient broadcasting of ops
- Provides efficient implementation of functions – arithmetic, statistic, trigonometric etc.

PRIME INTUIT
Finishing School

# Numpy Recap

- NumPy is missing features to enable data analysis on relational data like table.
  - No way to attach labels to data
  - No pre-built methods to fill missing values eg: fill 0 in NaN
  - No way to group data
  - No way to pivot data

- Pandas is built on top of NumPy to make data processing on relational data easier
- For a Data Scientist working in Python, Pandas is a crucial tool Ingesting, Storing, Pre-processing, Summarizing and Visualizing data can all be done effectively with Pandas