

CONFIDENTIAL &amp; RESTRICTED

Gaurav Gupta

# Numpy

- Importing
- Numpy Array: Indexing, Slicing, Reshaping
- Numpy Functions

tuteur.py@gmail.com

CONFIDENTIAL &amp; RESTRICTED

Gaurav Gupta

## What is Numpy

---

**NumPy** is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform, and random number capabilities

*Source: [numpy.org](http://numpy.org)*

tuteur.py@gmail.com

CONFIDENTIAL &amp; RESTRICTED

Gaurav Gupta

## Numpy import and numpy array

---

- **import numpy as np**
- Uses a datatype: **ndarray**
- Data in these arrays is homogeneous.
- Can be created via the following function:
  - `array(<some sequence>, dtype):`

tuteur.py@gmail.com

CONFIDENTIAL &amp; RESTRICTED

Gaurav Gupta

## Data Types

---

- `np.int8, np.int16, np.int32, np.int64`
- `np.uint8, np.uint16, np.uint32, np.uint64`
- `np.float32, np.float64`

*Source: [numpy.org](https://numpy.org)*

tuteur.py@gmail.com

CONFIDENTIAL &amp; RESTRICTED

Gaurav Gupta

## More on Data types

---

- '?' *boolean*
- 'b' *(signed) byte*
- 'B' *unsigned byte*
- 'i' *(signed) integer*
- 'u' *unsigned integer*
- 'f' *floating-point*
- 'c' *complex-floating point*
- 'M' *datetime*
- 'O' *(Python) objects*
- 'U' *Unicode string*

The first character specifies the kind of data and the remaining characters specify the number of bytes per item,

Source:  
<https://docs.scipy.org>

tuteur.py@gmail.com

CONFIDENTIAL &amp; RESTRICTED

Gaurav Gupta

## Array Creation

---

- `arange(start, end, step)`
- `Random.randint(start, end, size = <no of elements>)` # default gives one no.
- `linspace(start, end, count)`
- `zeros(shape)` # shape single arg or tuple of shape
- `ones(shape)`

tuteur.py@gmail.com

CONFIDENTIAL &amp; RESTRICTED

Gaurav Gupta

## Numpy Array slicing-indexing

---

- Slicing works similar to normal lists

`array[ <row index/slice>, <column index/slice> ]`

Ex:

`array[ 1:4, [3,4] ]`

- For multidimensional slicing use the comma syntax:

`array[ dim1, dim2, dim3, ..... ]`

tuteur.py@gmail.com

CONFIDENTIAL &amp; RESTRICTED

Gaurav Gupta

## Numpy Operations and Aggregate Functions

---

- Supports: \*, /, -, % etc. Behavior depends on operands
- Relations Operations return matrices of Booleans which can be used as indexes  
>, <, >=, <=
- Min, max, sum, mean
- These take **axis** as argument which denotes 0 (column wise), 1(row wise) and so forth for multiple dimensions.
- `any()`, `all()`, `np.isnan( <array> )`

tuteur.py@gmail.com

CONFIDENTIAL &amp; RESTRICTED

Gaurav Gupta

## Numpy Filtering out data

---

- Use the **where** function
- Where takes a Boolean matrix as argument and returns a two arrays containing row and column indexes.
- This can be used as an index into the original array.
- *`index = np.where( (array > x) & (array < y) )`*  
*`elements = array(index)`*

tuteur.py@gmail.com

CONFIDENTIAL &amp; RESTRICTED

Gaurav Gupta

## Looping and Stacking

---

- Use `vstack` or `hstack`.
- The dimensions must be aligned before stacking can be done.

tuteur.py@gmail.com

CONFIDENTIAL & RESTRICTED

Gaurav Gupta

# Pandas

- Importing
- Series and DataFrame

tuteur.py@gmail.com

CONFIDENTIAL & RESTRICTED

Gaurav Gupta

## Import and Series and DataFrame

- **import pandas as pd**
- Data Types:
  - Series(1-D) : Homogeneous. Behaves like a list
  - DataFrame(2-D) : Heterogeneous Columns

tuteur.py@gmail.com

CONFIDENTIAL &amp; RESTRICTED

Gaurav Gupta

## Series and DataFrame

---

- `Series(<sequence>, index, dtype)`
  - `Series([1, 3, 5, np.nan, 6, 8])`
- `DataFrame (<data object>, [index=], [columns=])`
  - `df = pd.DataFrame( np.random.randn(6, 4))`
  - `pd.DataFrame( {'A': 1,...:'C': pd.Series(1, index=list(range(4)), dtype='float32'),  
'D': np.array([3]*4, dtype='int32'),  
'E': pd.Categorical(["test", "train", "test", "train"]),  
'F': 'foo'})`

tuteur.py@gmail.com

CONFIDENTIAL &amp; RESTRICTED

Gaurav Gupta

## Data Frame attributes and viewing data

---

- Attributes:
  - `shape`,
  - `dtypes`
  - `columns`
  - `index`
- Viewing:
  - `head()` : view top few rows
  - `tail()` : view last few rows
  - `describe()` : view stats about the data

tuteur.py@gmail.com

CONFIDENTIAL &amp; RESTRICTED

Gaurav Gupta

## DataFrame slicing, indexing or viewing

---

- `df[ <index> ]` : gives a column
- `df[ start: end ]` : slices on rows
- To index or slice rows and columns use *loc* or *iloc*.
- `loc[start:end, ....]` : slices on basis of  
slices on row and column label; indexes can be a list as well.
- `iloc[start: end, start:end]`  
`iloc[ < list of row indexes>, <list of column indexes>]`  
slices via indexes

tuteur.py@gmail.com

CONFIDENTIAL &amp; RESTRICTED

Gaurav Gupta

## Operations and Aggregate Functions

---

- Supports: `*`, `/`, `-`, `%` etc.  
Behavior depends on operands
- Relations Operations return matrices of Booleans which can be used as indexes  
`>`, `<`, `>=`, `<=`
- Min, max, sum, mean
- These take **axis** as argument which denotes 0 (column wise), 1(row wise) and so forth for multiple dimensions.

tuteur.py@gmail.com



CONFIDENTIAL &amp; RESTRICTED

Gaurav Gupta

## Functions

---

- `<series>.value_counts()`  
Returns frequency of each type in the Series
- `apply(lambda)`  
applies the lambda of function to a row or a column  
similar: `applymap` (acts elementwise)
- `replace()`  
Replace data, using dictionary, list of original and new values or regex
- `rename()` : Rename row and column indexes

tuteur.py@gmail.com

CONFIDENTIAL &amp; RESTRICTED

Gaurav Gupta

## Checking NaN Functions

---

- `isna()`
- `fillna()`
- `dropna()`
- `isnull()`
- `any()`
- `all()`

tuteur.py@gmail.com

CONFIDENTIAL &amp; RESTRICTED

Gaurav Gupta

## Grouping and sorting

---

- `groupby( <list of columns> )`  
    `get_group( <tuple of filter values> )`
- Once grouped, functions can be applied on the grouped column labels

tuteur.py@gmail.com

CONFIDENTIAL &amp; RESTRICTED

Gaurav Gupta

## Importing and Dumping Datasets

---

- `read_csv(<file name>)`
- `read_excel(<file name>, skiprows=<number>, sheet_name=<name of sheet>)`
- `to_csv(<file name>, [index=True/False] )`
- `to_excel()`                      # requires `pd.ExcelWriter`

tuteur.py@gmail.com