Tools in Scientific Computing

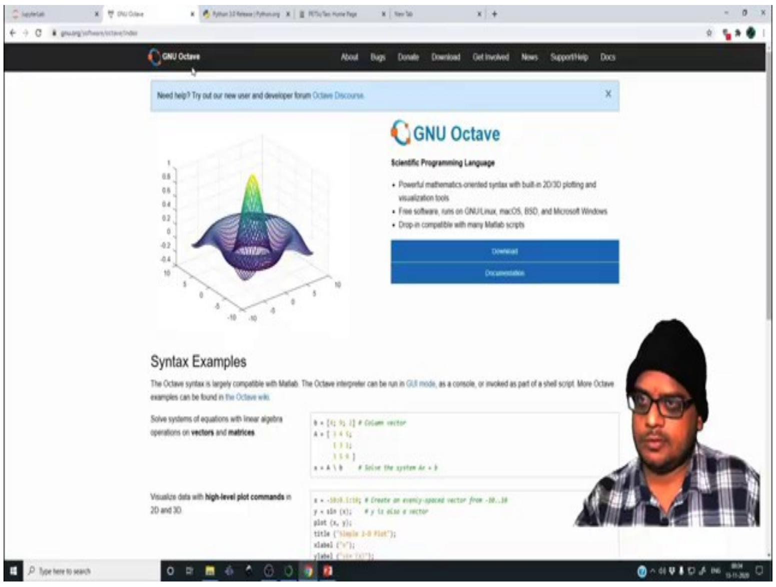
Prof. Aditya Bandopadhyay
Department of Mechanical Engineering
Indian Institute of Technology, Kharagpur

Lecture – 01
Preliminaries and Data Types

Hello and welcome to this first lecture. In this particular lecture, we are going to look at some preliminaries and some data types that we will be using in this particular course. And these data types are quite useful, even if you are doing something unrelated to things that we will discuss in this course. But just a first word of caution, this is not a course on explaining the different physical processes or the different mathematics behind the various phenomena that we are going to discuss.

However, we are going to try to keep it as self-contained as possible. We are going to look at some background some theory and then, we will look at the particular code. But all said and done, it will give you the confidence to tackle various problems on your own.

(Refer Slide Time: 01:19)



Need help? Try out our new user and developer forum Octave Discourse.

GNU Octave

Scientific Programming Language

- Powerful mathematics-oriented syntax with built-in 2D/3D plotting and visualization tools
- Free software, runs on GNU/Linux, macOS, BSD, and Microsoft Windows
- Drop-in compatible with many Matlab scripts

Syntax Examples

The Octave syntax is largely compatible with Matlab. The Octave interpreter can be run in GUI mode, as a console, or invoked as part of a shell script. More Octave examples can be found in the Octave wiki.

Solve systems of equations with linear algebra operations on **vectors** and **matrices**.

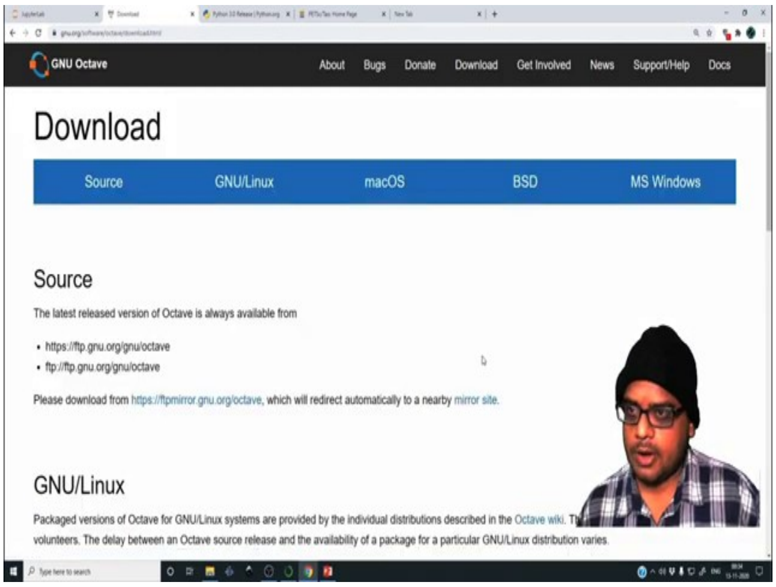
```
b = [4; 9; 2] # Column vector
A = [ 3 4 5;
      1 3 1;
      3 5 9 ]
x = A \ b     # Solve the system Ax = b
```

Visualize data with **high-level plot commands** in 2D and 3D.

```
x = -10:0.1:10; # Create an evenly-spaced vector from -10..10
y = sin (x);    # y is also a vector
plot (x, y);
title ("Simple 2-D Plot");
xlabel ("x");
ylabel ("sin (x)");
```

And that is what so first thing is. First, you are going to have to install GNU Octave. So, just go to the website gnu dot org slash software slash octave slash index and you are going to end up on this page. You can simply google, GNU-Octave; you can download this.

(Refer Slide Time: 01:39)



Download

Source | GNU/Linux | macOS | BSD | MS Windows

Source

The latest released version of Octave is always available from

- https://ftp.gnu.org/gnu/octave
- ftp://ftp.gnu.org/gnu/octave

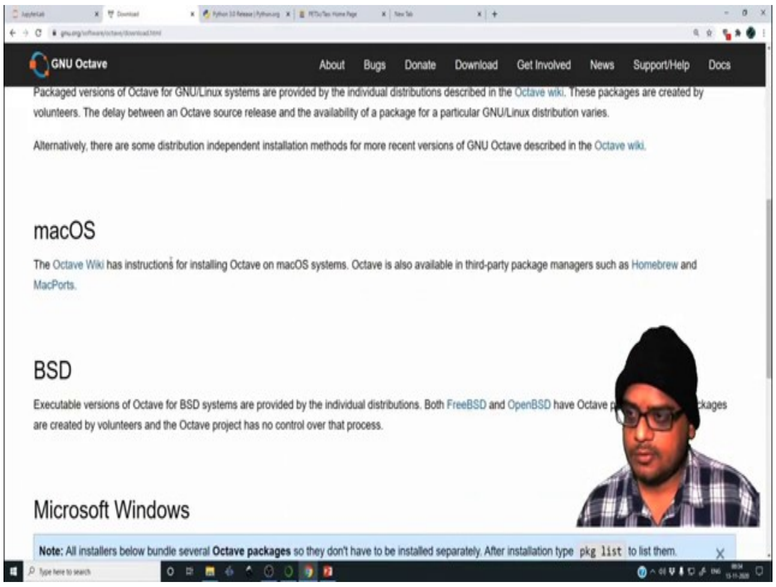
Please download from https://ftpmirror.gnu.org/octave, which will redirect automatically to a nearby mirror site.

GNU/Linux

Packaged versions of Octave for GNU/Linux systems are provided by the individual distributions described in the Octave wiki. Th
volunteers. The delay between an Octave source release and the availability of a package for a particular GNU/Linux distribution varies.

Once you click on download, you get an option of what operating system you are working with; GNU Linux, macOS or Windows.

(Refer Slide Time: 01:50)



Packaged versions of Octave for GNU/Linux systems are provided by the individual distributions described in the Octave wiki. These packages are created by volunteers. The delay between an Octave source release and the availability of a package for a particular GNU/Linux distribution varies.

Alternatively, there are some distribution independent installation methods for more recent versions of GNU Octave described in the Octave wiki.

macOS

The Octave Wiki has instructions for installing Octave on macOS systems. Octave is also available in third-party package managers such as Homebrew and MacPorts.

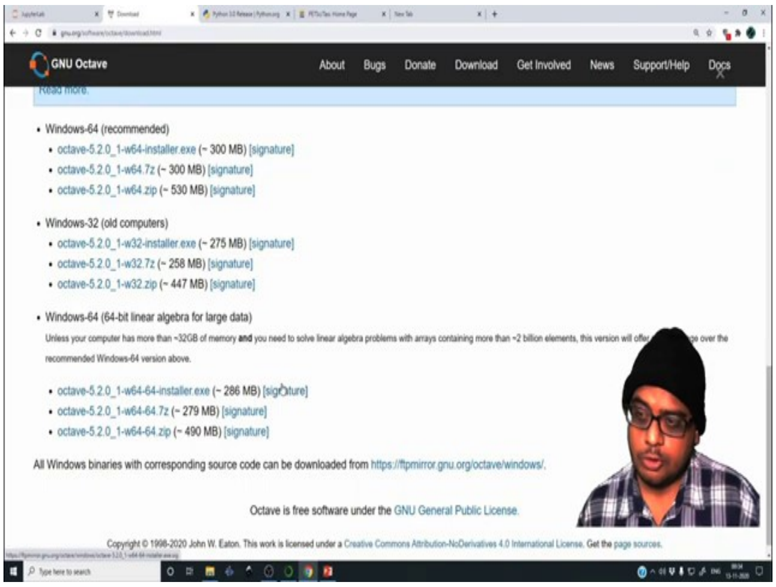
BSD

Executable versions of Octave for BSD systems are provided by the individual distributions. Both FreeBSD and OpenBSD have Octave p kages are created by volunteers and the Octave project has no control over that process.

Microsoft Windows

Note: All installers below bundle several **Octave packages** so they don't have to be installed separately. After installation type `pkg list` to list them.

(Refer Slide Time: 01:54)



Read more.

- Windows-64 (recommended)
  - octave-5.2.0_1-w64-installer.exe (~ 300 MB) [signature]
  - octave-5.2.0_1-w64.7z (~ 300 MB) [signature]
  - octave-5.2.0_1-w64.zip (~ 530 MB) [signature]
- Windows-32 (old computers)
  - octave-5.2.0_1-w32-installer.exe (~ 275 MB) [signature]
  - octave-5.2.0_1-w32.7z (~ 258 MB) [signature]
  - octave-5.2.0_1-w32.zip (~ 447 MB) [signature]
- Windows-64 (64-bit linear algebra for large data)

  Unless your computer has more than ~32GB of memory **and** you need to solve linear algebra problems with arrays containing more than ~2 billion elements, this version will offer [illegible] over the recommended Windows-64 version above.

  - octave-5.2.0_1-w64-64-installer.exe (~ 286 MB) [signature]
  - octave-5.2.0_1-w64-64.7z (~ 279 MB) [signature]
  - octave-5.2.0_1-w64-64.zip (~ 490 MB) [signature]

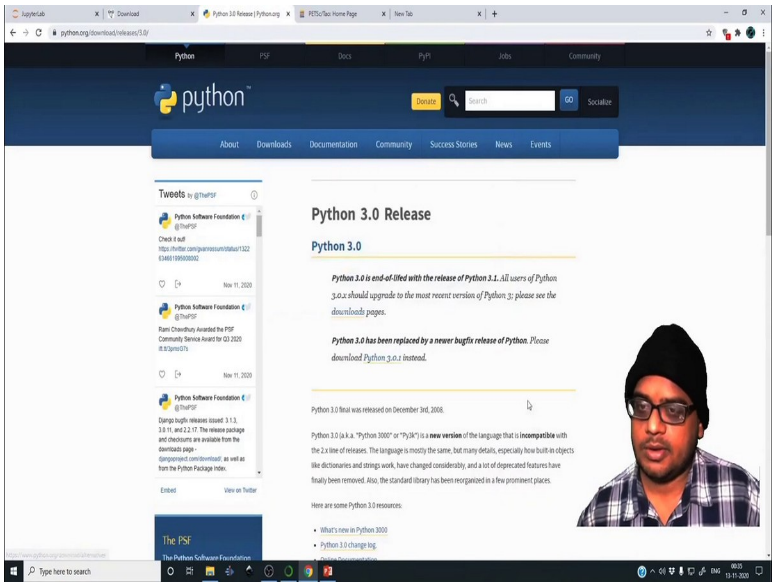
All Windows binaries with corresponding source code can be downloaded from https://ftpmirror.gnu.org/octave/windows/.

Octave is free software under the GNU General Public License.

Copyright © 1998-2020 John W. Eaton. This work is licensed under a Creative Commons Attribution-NoDerivatives 4.0 International License. Get the page sources.

Now, in this particular case, I am working on windows. So, you can go all the way to windows and you can choose a 64-bit installer or 32, if you are using an older computer and if you intend to use GNU Octave for handling very large matrices, you can install this particular package. But be careful, you need to have a sufficient amount of RAM in order for that particular version to work.

(Refer Slide Time: 02:21)



Python Software Foundation @ThePSF
Check it out! https://twitter.com/gvanrossum/status/1322634661995008002
Nov 11, 2020

Python Software Foundation @ThePSF
Rami Chowdhury Awarded the PSF Community Service Award for Q3 2020 ift.tt/3pmsG7s
Nov 11, 2020

Python Software Foundation @ThePSF
Django bugfix releases issued: 3.1.3, 3.0.11, and 2.2.17. The release package and checksums are available from the downloads page - djangoproject.com/download/, as well as from the Python Package Index.

Python 3.0 Release

Python 3.0

***Python 3.0 is end-of-lifed with the release of Python 3.1.** All users of Python 3.0.x should upgrade to the most recent version of Python 3; please see the downloads pages.*

***Python 3.0 has been replaced by a newer bugfix release of Python.** Please download Python 3.0.1 instead.*

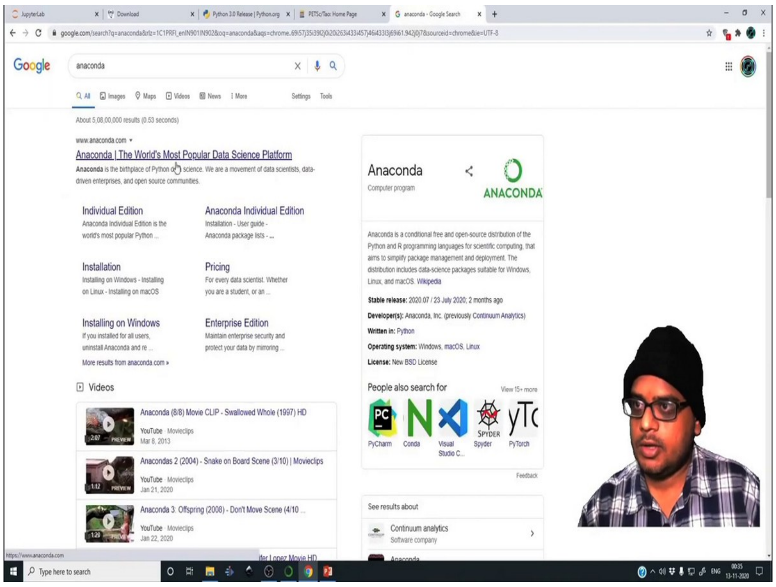
Python 3.0 final was released on December 3rd, 2008.

Python 3.0 (a.k.a. "Python 3000" or "Py3k") is a **new version** of the language that is **incompatible** with the 2.x line of releases. The language is mostly the same, but many details, especially how built-in objects like dictionaries and strings work, have changed considerably, and a lot of deprecated features have finally been removed. Also, the standard library has been reorganized in a few prominent places.

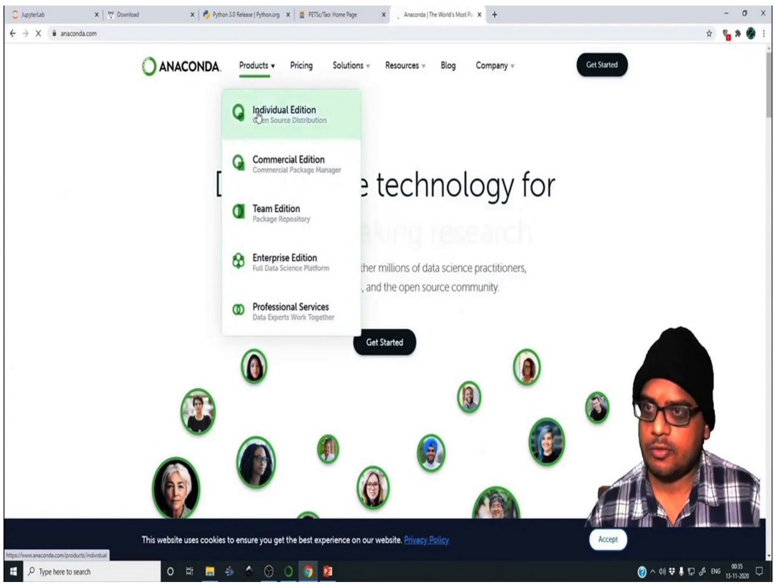
Here are some Python 3.0 resources:

- What's new in Python 3000
- Python 3.0 change log.

(Refer Slide Time: 02:31)

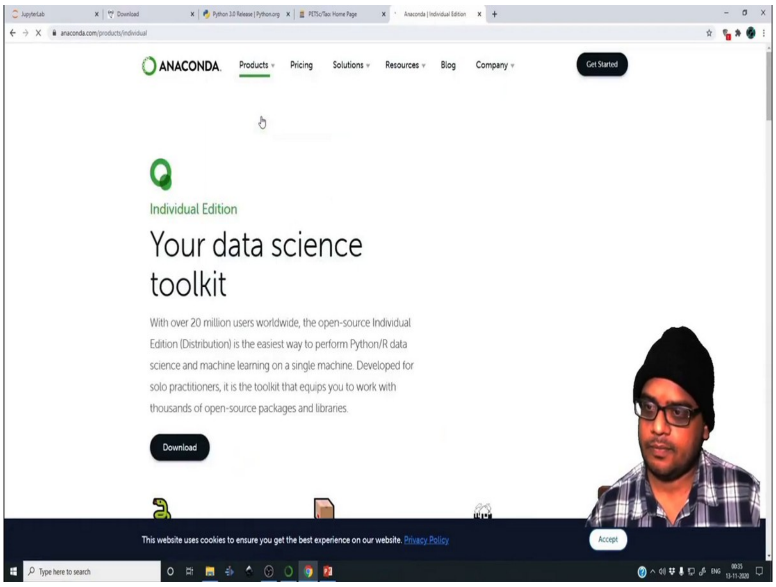


(Refer Slide Time: 02:34)



The other thing that you can install, I mean you can install Python 3. Yes, that is fine; but you can also install Anaconda. You can install anaconda and it will install everything in one go. So, you go to you Google anaconda; anaconda dot com, you go to products individual edition.

(Refer Slide Time: 02:42)



Individual Edition

Your data science toolkit

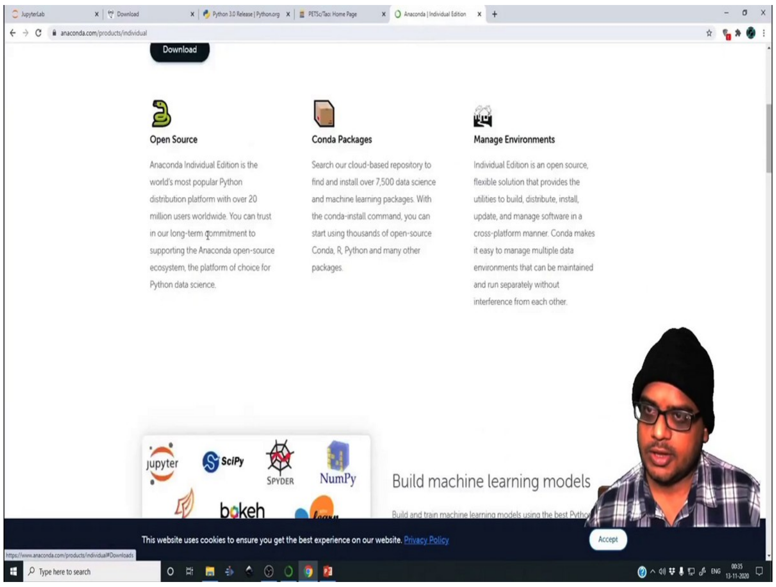
With over 20 million users worldwide, the open-source Individual Edition (Distribution) is the easiest way to perform Python/R data science and machine learning on a single machine. Developed for solo practitioners, it is the toolkit that equips you to work with thousands of open-source packages and libraries.

Download

This website uses cookies to ensure you get the best experience on our website. Privacy Policy

Accept

(Refer Slide Time: 02:49)



Download

Open Source

Anaconda Individual Edition is the world's most popular Python distribution platform with over 20 million users worldwide. You can trust in our long-term commitment to supporting the Anaconda open-source ecosystem, the platform of choice for Python data science.

Conda Packages

Search our cloud-based repository to find and install over 7,500 data science and machine learning packages. With the conda-install command, you can start using thousands of open-source Conda, R, Python and many other packages.

Manage Environments

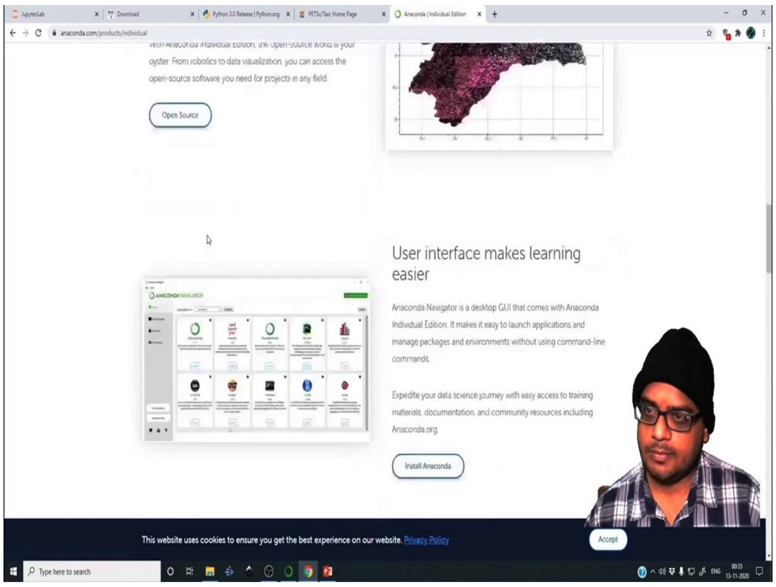
Individual Edition is an open source, flexible solution that provides the utilities to build, distribute, install, update, and manage software in a cross-platform manner. Conda makes it easy to manage multiple data environments that can be maintained and run separately without interference from each other.

Build machine learning models

This website uses cookies to ensure you get the best experience on our website. Privacy Policy

Accept

(Refer Slide Time: 02:49)


JupyterLab
Download
Python 3.0 Release | Python.org
PETSc/Tao: Home Page
Anaconda | Individual Edition
anaconda.com/products/individual
Open Source
Install Anaconda
Accept
Type here to search

With Anaconda Individual Edition, the open-source world is your oyster. From robotics to data visualization, you can access the open-source software you need for projects in any field.

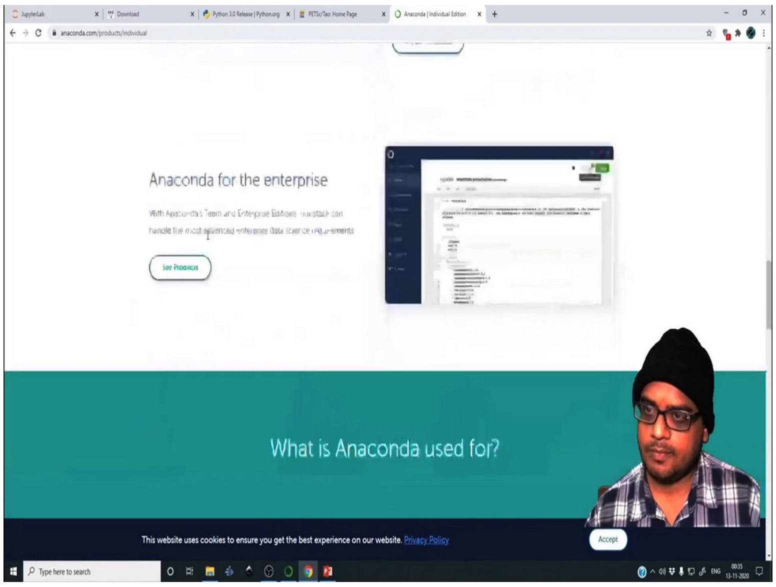
User interface makes learning easier

Anaconda Navigator is a desktop GUI that comes with Anaconda Individual Edition. It makes it easy to launch applications and manage packages and environments without using command-line commands.

Expedite your data science journey with easy access to training materials, documentation, and community resources including Anaconda.org.

This website uses cookies to ensure you get the best experience on our website. Privacy Policy

(Refer Slide Time: 02:49)


JupyterLab
Download
Python 3.0 Release | Python.org
PETSc/Tao: Home Page
Anaconda | Individual Edition
anaconda.com/products/individual
See Products
Accept
Type here to search

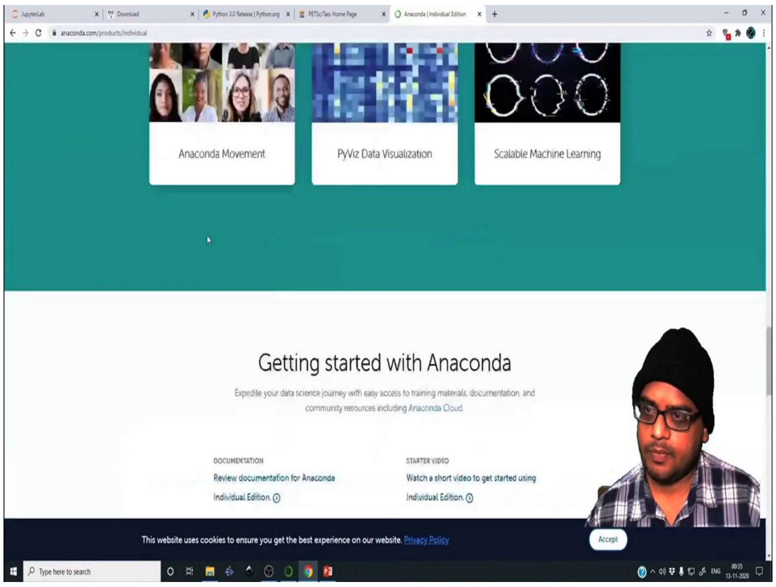
Anaconda for the enterprise

With Anaconda's Team and Enterprise Editions, your stack can handle the most advanced enterprise data science requirements

What is Anaconda used for?

This website uses cookies to ensure you get the best experience on our website. Privacy Policy

(Refer Slide Time: 02:49)



Getting started with Anaconda

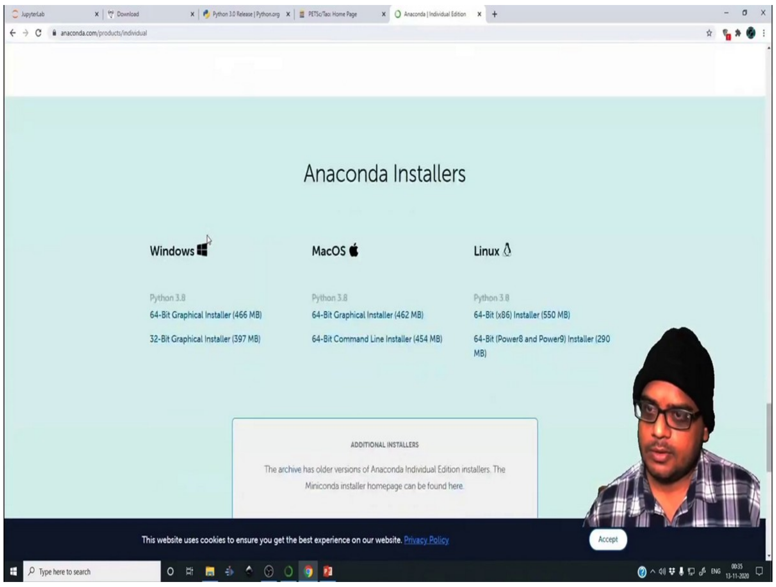
Expedite your data science journey with easy access to training materials, documentation, and community resources including Anaconda Cloud.

DOCUMENTATION
Review documentation for Anaconda Individual Edition.

STARTER VIDEO
Watch a short video to get started using Individual Edition.

This website uses cookies to ensure you get the best experience on our website. Privacy Policy

(Refer Slide Time: 02:50)



Anaconda Installers

Windows	MacOS	Linux
Python 3.8	Python 3.8	Python 3.8
64-Bit Graphical Installer (466 MB)	64-Bit Graphical Installer (462 MB)	64-Bit (x86) Installer (550 MB)
32-Bit Graphical Installer (397 MB)	64-Bit Command Line Installer (454 MB)	64-Bit (Power8 and Power9) Installer (290 MB)

ADDITIONAL INSTALLERS

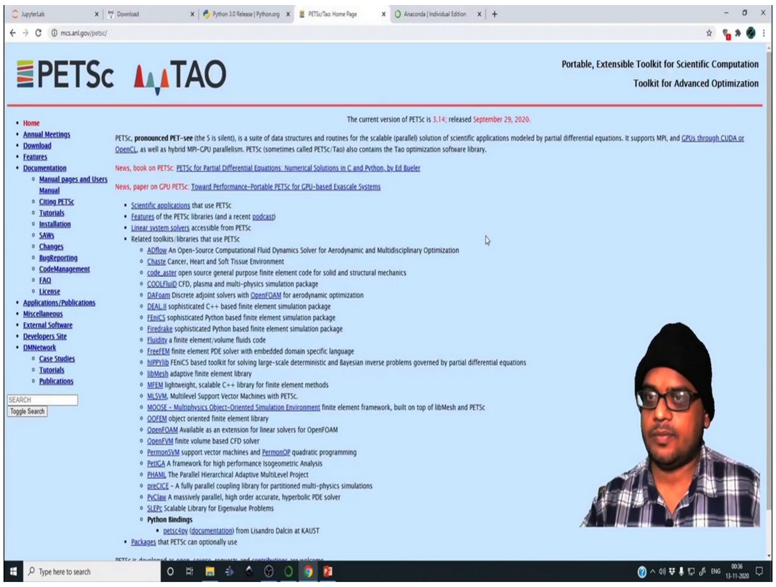
The archive has older versions of Anaconda Individual Edition installers. The Miniconda installer homepage can be found here.

This website uses cookies to ensure you get the best experience on our website. Privacy Policy

And then, you do a download. It will install everything; just follow all the instructions; so, if you are using windows, just install, just download the 64 bit graphical installer and make sure you follow all the instructions.

In particular, there is an instruction, you have to deselect the addition of the path to the environment variable. So, once you install this, you will have installed all the different packages that we are going to use in this particular course and in particular, you will have a full access to all the libraries through a very easy installation framework.

(Refer Slide Time: 03:26)



PETSc TAO

Portable, Extensible Toolkit for Scientific Computation

Toolkit for Advanced Optimization

- Home
- Annual Meetings
- Download
- Features
- Documentation
  - Manual pages and Users Manual
  - Citing PETSc
  - Tutorials
  - Installation
  - SAWs
  - Changes
  - BugReporting
  - CodeManagement
  - FAQ
  - License
- Applications/Publications
- Miscellaneous
- External Software
- Developers Site
- DMNetwork
  - Case Studies
  - Tutorials
  - Publications

The current version of PETSc is 3.14; released September 29, 2020.

PETSc, **pronounced PET-see** (the S is silent), is a suite of data structures and routines for the scalable (parallel) solution of scientific applications modeled by partial differential equations. It supports MPI, and GPUs through CUDA or OpenCL, as well as hybrid MPI-GPU parallelism. PETSc (sometimes called PETSc/Tao) also contains the Tao optimization software library.

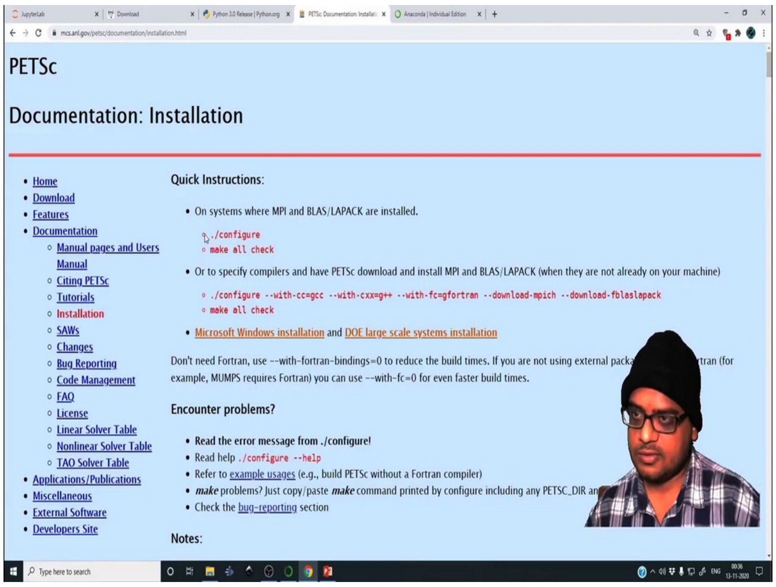
News, book on PETSc: PETSc for Partial Differential Equations: Numerical Solutions in C and Python, by Ed Bueler

News, paper on GPU PETSc: Toward Performance-Portable PETSc for GPU-based Exascale Systems

- Scientific applications that use PETSc
- Features of the PETSc libraries (and a recent podcast)
- Linear system solvers accessible from PETSc
- Related toolkits/libraries that use PETSc
  - ADflow An Open-Source Computational Fluid Dynamics Solver for Aerodynamic and Multidisciplinary Optimization
  - Chaste Cancer, Heart and Soft Tissue Environment
  - code_aster open source general purpose finite element code for solid and structural mechanics
  - COOLFluiD CFD, plasma and multi-physics simulation package
  - DAFoam Discrete adjoint solvers with OpenFOAM for aerodynamic optimization
  - DEAL.II sophisticated C++ based finite element simulation package
  - FEniCS sophisticated Python based finite element simulation package
  - Firedrake sophisticated Python based finite element simulation package
  - Fluidity a finite element/volume fluids code
  - FreeFEM finite element PDE solver with embedded domain specific language
  - hIPPYlib FEniCS based toolkit for solving large-scale deterministic and Bayesian inverse problems governed by partial differential equations
  - libMesh adaptive finite element library
  - MFEM lightweight, scalable C++ library for finite element methods
  - MLSVM, Multilevel Support Vector Machines with PETSc.
  - MOOSE - Multiphysics Object-Oriented Simulation Environment finite element framework, built on top of libMesh and PETSc
  - OOFEM object oriented finite element library
  - OpenFOAM Available as an extension for linear solvers for OpenFOAM
  - OpenFVM finite volume based CFD solver
  - PermonSVM support vector machines and PermonQP quadratic programming
  - PetIGA A framework for high performance Isogeometric Analysis
  - PHAML The Parallel Hierarchical Adaptive MultiLevel Project
  - preCICE - A fully parallel coupling library for partitioned multi-physics simulations
  - PyClaw A massively parallel, high order accurate, hyperbolic PDE solver
  - SLEPc Scalable Library for Eigenvalue Problems
  - **Python Bindings**
    - petsc4py (documentation) from Lisandro Dalcin at KAUST
- Packages that PETSc can optionally use

Lastly, you can install PETSc. For that, you need to first have a working GCC compiler. GCC is a set of compilers released by the GNU group.

(Refer Slide Time: 03:41)



PETSc

Documentation: Installation

- Home
- Download
- Features
- Documentation
  - Manual pages and Users Manual
  - Citing PETSc
  - Tutorials
  - Installation
  - SAWs
  - Changes
  - Bug Reporting
  - Code Management
  - FAQ
  - License
  - Linear Solver Table
  - Nonlinear Solver Table
  - TAO Solver Table
- Applications/Publications
- Miscellaneous
- External Software
- Developers Site

Quick Instructions:

- On systems where MPI and BLAS/LAPACK are installed.

```
./configure
make all check
```

- Or to specify compilers and have PETSc download and install MPI and BLAS/LAPACK (when they are not already on your machine)

```
./configure --with-cc=gcc --with-cxx=g++ --with-fc=gfortran --download-mpich --download-fblaslapack
make all check
```

- Microsoft Windows installation and DOE large scale systems installation

Don't need Fortran, use --with-fortran-bindings=0 to reduce the build times. If you are not using external packa tran (for example, MUMPS requires Fortran) you can use --with-fc=0 for even faster build times.

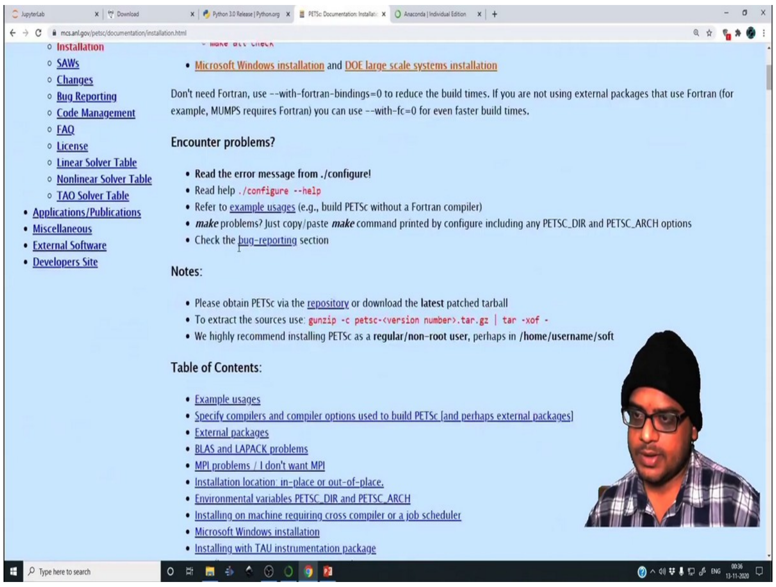
Encounter problems?

- **Read the error message from ./configure!**
- Read help `./configure --help`
- Refer to example usages (e.g., build PETSc without a Fortran compiler)
- ***make*** problems? Just copy/paste ***make*** command printed by configure including any PETSC_DIR an
- Check the bug-reporting section

Notes:

So, you go to installation. So, once you download it, you can install it using dot slash configure and then, make all ok. It is not a very complicated installation; but yes, it does take a bit of time.

(Refer Slide Time: 03:53)



- Installation
- SAWs
- Changes
- Bug Reporting
- Code Management
- FAQ
- License
- Linear Solver Table
- Nonlinear Solver Table
- TAO Solver Table
- Applications/Publications
- Miscellaneous
- External Software
- Developers Site

- Microsoft Windows installation and DOE large scale systems installation

Don't need Fortran, use --with-fortran-bindings=0 to reduce the build times. If you are not using external packages that use Fortran (for example, MUMPS requires Fortran) you can use --with-fc=0 for even faster build times.

Encounter problems?

- **Read the error message from ./configure!**
- Read help `./configure --help`
- Refer to example usages (e.g., build PETSc without a Fortran compiler)
- ***make*** problems? Just copy/paste ***make*** command printed by configure including any PETSC_DIR and PETSC_ARCH options
- Check the bug-reporting section

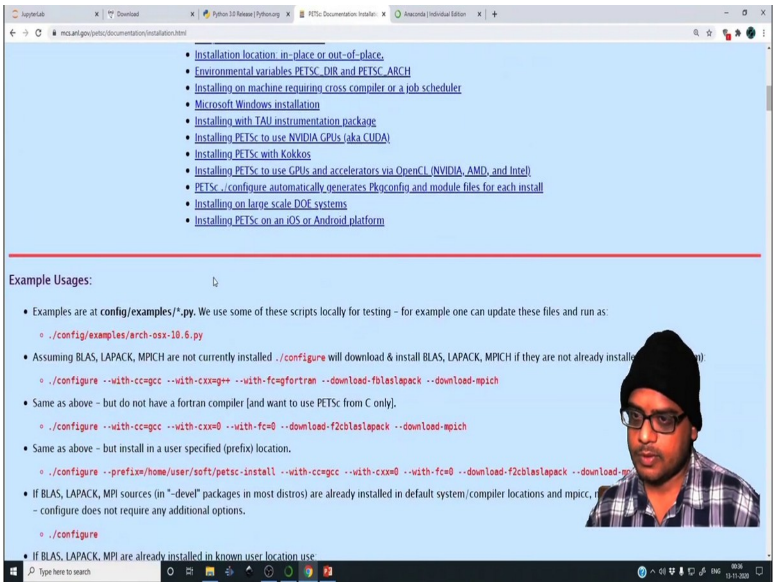
Notes:

- Please obtain PETSc via the repository or download the **latest** patched tarball
- To extract the sources use: `gunzip -c petsc-<version number>.tar.gz | tar -xof -`
- We highly recommend installing PETSc as a **regular/non-root user**, perhaps in **/home/username/soft**

Table of Contents:

- Example usages
- Specify compilers and compiler options used to build PETSc [and perhaps external packages]
- External packages
- BLAS and LAPACK problems
- MPI problems / I don't want MPI
- Installation location: in-place or out-of-place.
- Environmental variables PETSC_DIR and PETSC_ARCH
- Installing on machine requiring cross compiler or a job scheduler
- Microsoft Windows installation
- Installing with TAU instrumentation package

(Refer Slide Time: 03:55)



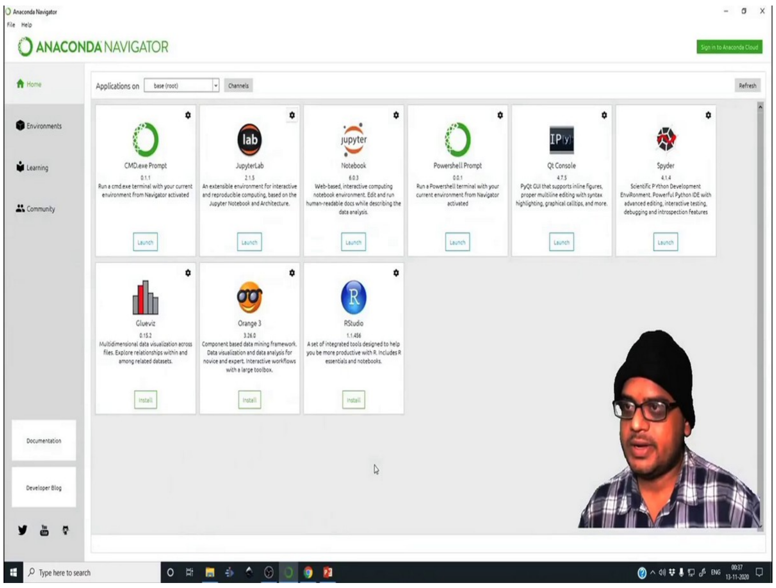
- Installation location: in-place or out-of-place.
- Environmental variables PETSC_DIR and PETSC_ARCH
- Installing on machine requiring cross compiler or a job scheduler
- Microsoft Windows installation
- Installing with TAU instrumentation package
- Installing PETSc to use NVIDIA GPUs (aka CUDA)
- Installing PETSc with Kokkos
- Installing PETSc to use GPUs and accelerators via OpenCL (NVIDIA, AMD, and Intel)
- PETSc ./configure automatically generates Pkgconfig and module files for each install
- Installing on large scale DOE systems
- Installing PETSc on an iOS or Android platform

Example Usages:

- Examples are at **config/examples/*.py.** We use some of these scripts locally for testing - for example one can update these files and run as:
  - `./config/examples/arch-osx-10.6.py`
- Assuming BLAS, LAPACK, MPICH are not currently installed `./configure` will download & install BLAS, LAPACK, MPICH if they are not already installe n):
  - `./configure --with-cc=gcc --with-cxx=g++ --with-fc=gfortran --download-fblaslapack --download-mpich`
- Same as above - but do not have a fortran compiler [and want to use PETSc from C only].
  - `./configure --with-cc=gcc --with-cxx=0 --with-fc=0 --download-f2cblaslapack --download-mpich`
- Same as above - but install in a user specified (prefix) location.
  - `./configure --prefix=/home/user/soft/petsc-install --with-cc=gcc --with-cxx=0 --with-fc=0 --download-f2cblaslapack --download-mp`
- If BLAS, LAPACK, MPI sources (in "-devel" packages in most distros) are already installed in default system/compiler locations and mpicc, r - configure does not require any additional options.
  - `./configure`
- If BLAS, LAPACK, MPI are already installed in known user location use:

And it is much easier to install on Linux than on Windows. But regardless, whatever we are going to study is independent of the platform. Whatever I am going to show Octave, Python all that is available on all operating systems. Whatever you develop on Windows will run the same on Linux as well.

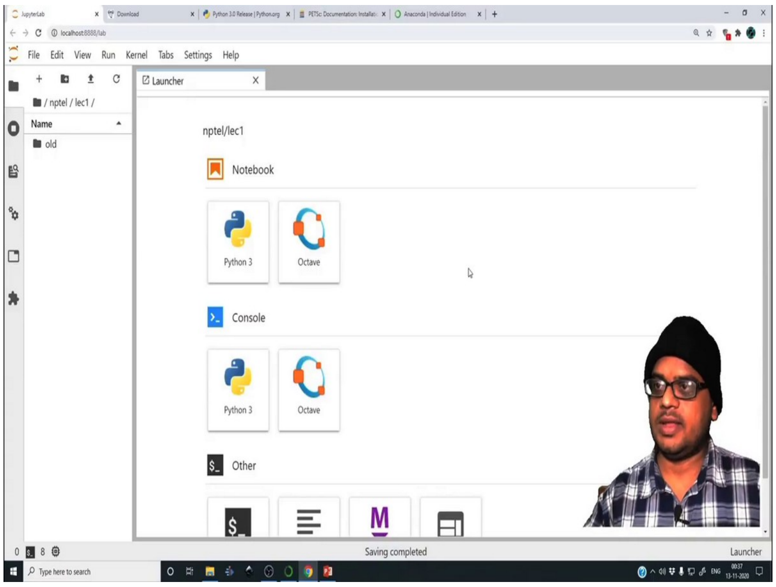
(Refer Slide Time: 04:22)



So, once you install anaconda, once you launch anaconda, you will be greeted by a screen which looks something like this. So, you have CMD command prompt, JupyterLab, Jupyter, Powershell, Qt, Spyder. So, Spyder is something which is quite useful.

So, it is a IDE; meaning, it is a integrated development environment in which we can write down scripts, much in the way you would write down scripts in MATLAB. There are some other packages as well. Once you click on Jupyter lab, it will pop up a window which would look something like this.

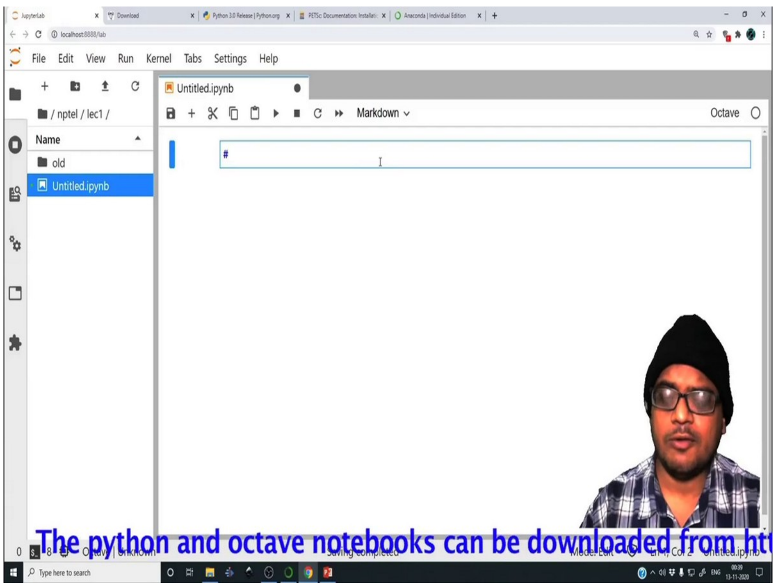
(Refer Slide Time: 04:58)



So, Jupyter lab is an environment in which you make notebooks. So, notebooks are file in which we can input text. So, let me just click on this. So, the beautiful thing about Jupyter lab is that you can make python programs as well as you can develop octave programs.

So, basically it is a front end of these kinds of kernels ok. These things are called as kernels and you can run whatever you want. So, I have added into my Jupyter lab these two kernels; Python 3 and Octave. By default, you will have Python 3 added; but if you want you can add, octave as a kernel to this as well. Make sure you have installed octave prior to installation of anaconda.

(Refer Slide Time: 05:52)



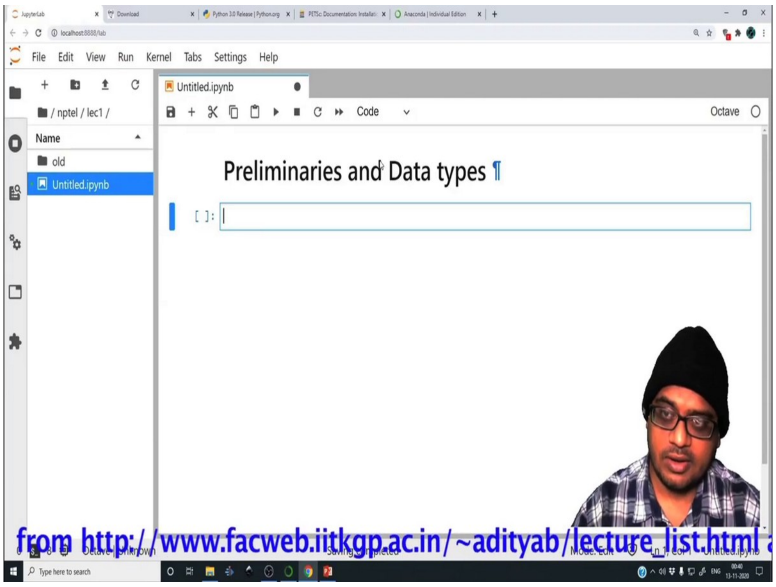
The python and octave notebooks can be downloaded from htt

Let me click on this. So, it would open a notebook style of file. So, a notebook is something in which you input text, you input mathematical equations, you input scripts of code, you can embed figures. So, it is a all-encompassing document which would contain all the things that one would need in order to encapsulate the entire idea.

You do not need to write down documentation separately and the program separately and the figure separately. You can embed everything in one go; once you are done, you can export the entire file as a PDF, as a word file, as a latex file whatever. So, these things are called as cells. Inside a cell, you can either encode a code or markdown.

So, markdown is a kind of mark-up language which is used to quickly write down text with headings, with equations, without really worrying about the formatting. In order to input mark down into the cell, I will select markdown. Let me put a title. So, this is a very well-defined syntax for creating titles, for creating bullet points in markdown. So, have a look at the markdown tutorial. I am not going to discuss about that and let us jump straight in into the programming part.

(Refer Slide Time: 07:26)

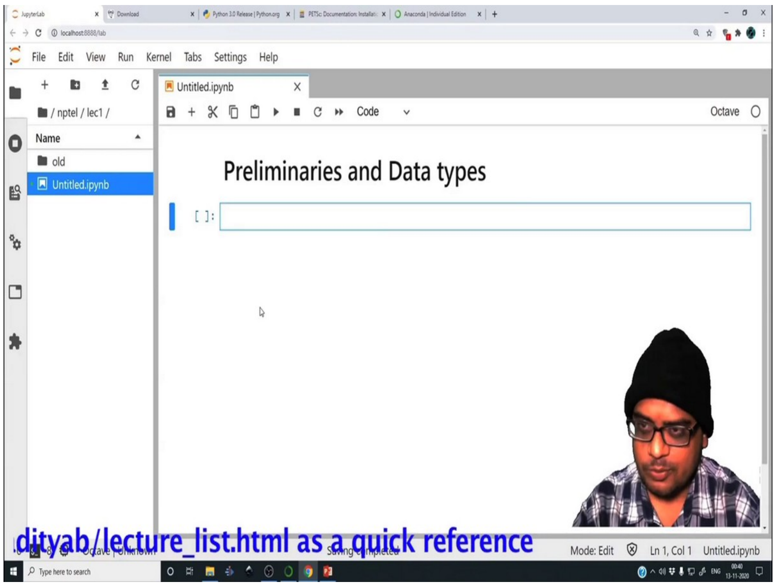


Preliminaries and Data types ¶

[ ]:

from http://www.facweb.iitkgp.ac.in/~adityab/lecture_list.html

(Refer Slide Time: 07:58)



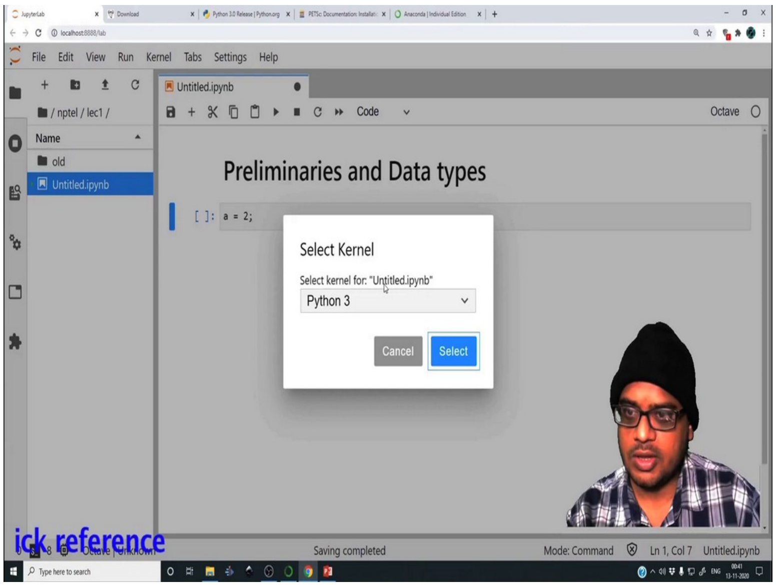
Preliminaries and Data types

[ ]:

dityab/lecture_list.html as a quick reference

So, let me create the title of this file. So, in order to execute a particular line, we would press shift enter; upon pressing shift enter, you see that the title has been converted into a heading ok. So, data types. So, we have already covered the preliminaries of how to install these packages; have a look at them, take some time to read through the various installation procedures. It is very simple to do ok. Let us jump straight into it.

(Refer Slide Time: 08:09)



Preliminaries and Data types

```
[ ]: a = 2;
```

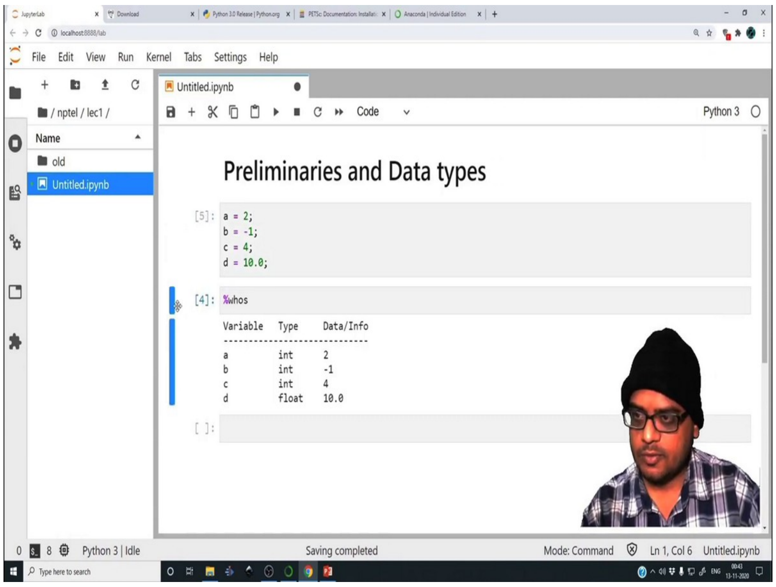
Select Kernel

Select kernel for: "Untitled.ipynb"

Python 3

The various data types for the first and the most-simplest data type will be that of a scalar. So, in python we can define a scalar directly. Let us define a=2. So, let me change the kernel to python ok.

(Refer Slide Time: 08:23)



Preliminaries and Data types

```
[5]: a = 2;
     b = -1;
     c = 4;
     d = 10.0;

[4]: %whos

Variable   Type    Data/Info
----------------------------
a          int     2
b          int     -1
c          int     4
d          float   10.0

[ ]:
```

So, a=2 fine. Let me press enter and keep on inputting. So, let me write b=- 1; c= 4 and d =10. So, now I have defined 4 variables; but I have not yet run the program. In order to run this particular cell, so things are encapsulated inside cells.

Note that this particular cell by default, a new cell is created as a code. Whatever you write inside a cell will be interpreted as a code. If you want to interpret it differently, I can change it to markdown. So, this will no longer be a code, this will be simply plain text; but I want it to be a code.

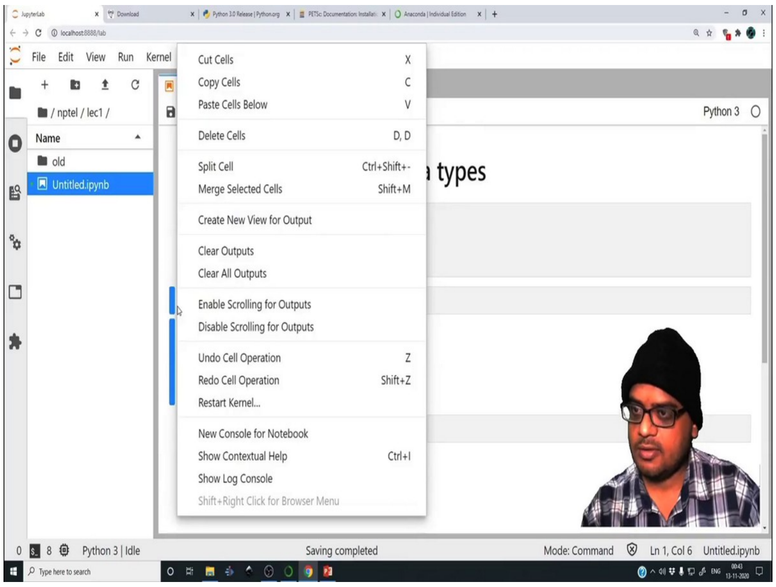
So, now I will press shift enter and now, it has loaded all the variables in memory. In order to check what variables, we have. We can put this kind of a magic command. So, Jupyter lab, this kind of an environment has different kinds of magic commands. So, this is one magic command. As we go in the course we will come across many other magic commands and you can Google it and find out what are the different magic commands.

So, let me press shift enter, it will show me that I have 4 variables. The data type is integer and the data is this. In fact, let me make d = 10.0; let me press shift enter to execute that cell once again, now let me press.

So, automatically you will see that python has declared d as a variable and its type is float and the value it is storing is 10.0 and this is somewhat different from declarations in a much more imperative programming language like C, in which you have to declare a priori what are the kind of variable will be.

So, in order to declare a as 2, you have to first write down 'int a' and then later on, you can say a =2 or you can simply write int a= 2. So, the same declaration in C would have been something like this; but this is not a course on C and we will continue on the style of python. So, very simple way of expressing your thoughts. So, now, I let me just delete this variable block.

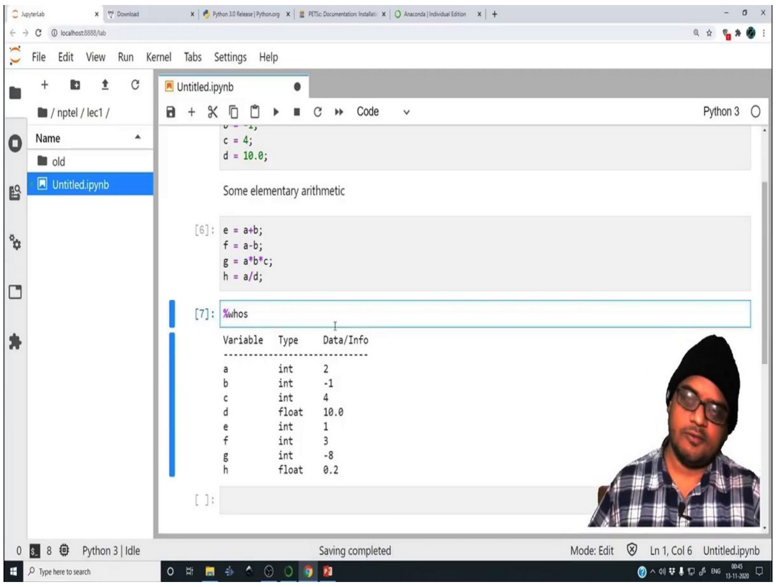
(Refer Slide Time: 10:54)



So, I can right click on this, I can say delete cells ok. Now, let us perform some arithmetic. So, I can press the escape key to focus on that cell, try it out; I can press enter to again come into that cell. So, I lose focus by pressing escape, I gain focus by pressing enter and the selected cell is shown by this blue border ok.

So, I press escape, I press M to convert the cell from code to markdown. Now, I will write over here. So, all you can do all of this with the help of all these the mouse things as well. But I prefer to do it with the help of keyboard shortcuts.

(Refer Slide Time: 11:44)



```
c = 4;
d = 10.0;
```

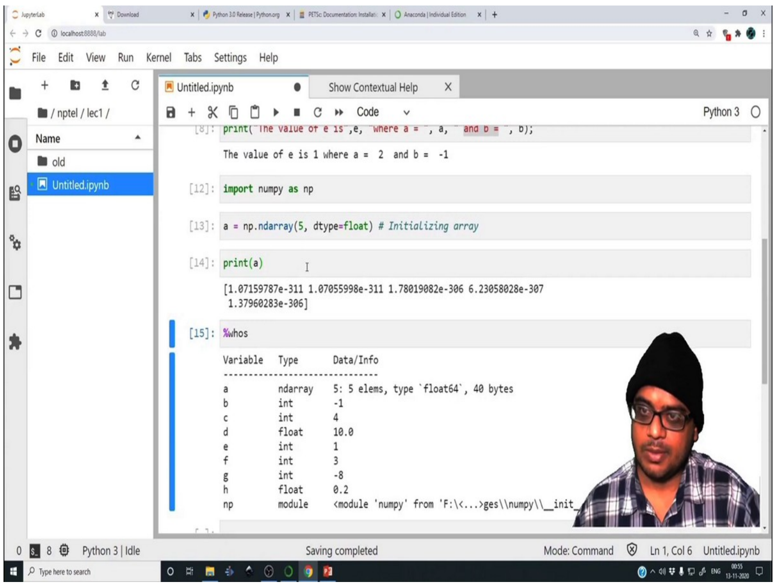
Some elementary arithmetic

```
[6]: e = a+b;
     f = a-b;
     g = a*b*c;
     h = a/d;

[7]: %whos

Variable   Type    Data/Info
----------------------------
a          int     2
b          int     -1
c          int     4
d          float   10.0
e          int     1
f          int     3
g          int     -8
h          float   0.2

[ ]:
```

So, let me write down here, some elementary arithmetic shift enter. So, now, that is interpreted as text; it is not a code, it is simply a text ok. So, let me write down e=a + b; f = a - b; g = a * b * c and h =a /d.

I can execute this, there is no error. If there was an error, it would has come right below this particular cell. Now, suppose I want to print out all the variables; of course, I can do whos again and it will show me all the variables that we have. So, e = a + b, 2 -1=1, f = a - b. So, 2 --1 is 3; we can verify all these simple things.

(Refer Slide Time: 12:50)



```
[8]: print("The value of e is",e, "where a = ", a, " and b = ", b);

     The value of e is 1 where a =  2  and b =  -1

[12]: import numpy as np

[13]: a = np.ndarray(5, dtype=float) # Initializing array

[14]: print(a)

      [1.07159787e-311 1.07055998e-311 1.78019082e-306 6.23058028e-307
       1.37960283e-306]

[15]: %whos

      Variable   Type       Data/Info
      -------------------------------
      a          ndarray    5: 5 elems, type `float64`, 40 bytes
      b          int        -1
      c          int        4
      d          float      10.0
      e          int        1
      f          int        3
      g          int        -8
      h          float      0.2
      np         module     <module 'numpy' from 'F:\<...>ges\\numpy\\__init_
```

But another way of simply printing out a given variable is writing this print within double quotes. So, we can write down the value of e = a +b or e is and we can write over here e, where a = a and b = b. Let me execute this. Upon execution, you will see, I did as printed out this particular string and wherever we have put down the variable, it will output the value of the particular variable.

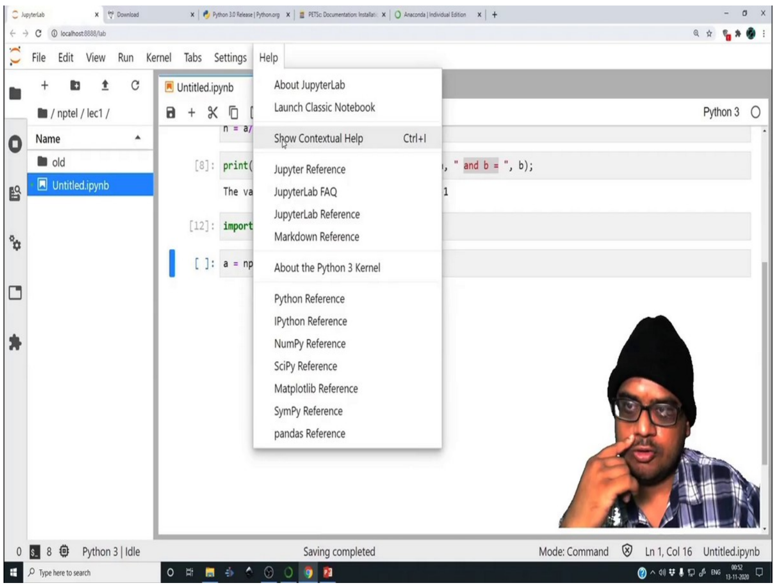
So, for example, the value of e is, so this is printed verbatim. So, this is printed verbatim, while this is the value of this particular variable. This particular thing is printed verbatim, then we print the value of a, then this is verbatim and then, we print the value of b.

So, this is how you can put down meaningful print statements, in order to make more sense of your code and this is especially important when you are debugging certain codes, so debugging the old school style ok. So, this is a very simple arithmetic operation on scalars and we have also shown how we can print out the variables. Let us now move into how we can declare arrays. So, there is a very special module in python, it is called as numpy. So, we have to first import that particular module.

So, we will write import numpy as np. So, the particular phrase as np implies that np is an alias for the packages inside the module numpy. Python has a lot of modules one of which is numpy, there are several other modules called as scipy or opencv. So, opencv is a module for image processing; scipy is a package for scientific computations; numpy is a package for numerical methods and data structures.

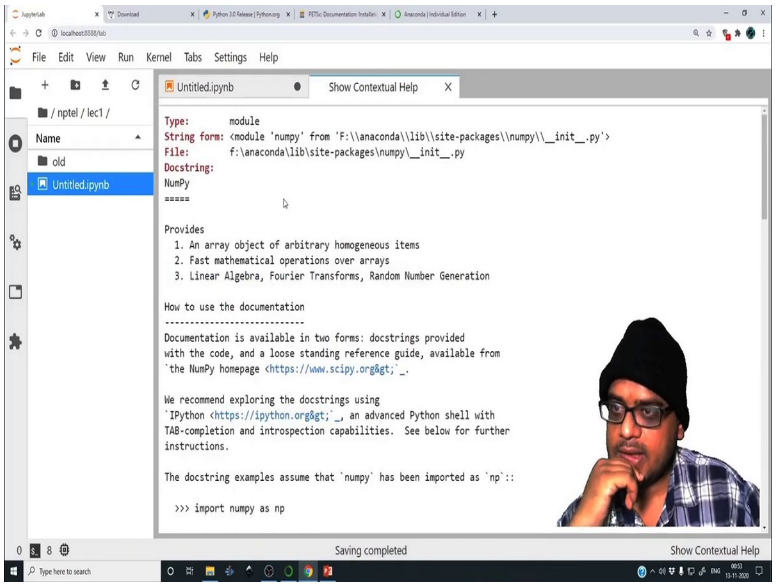
So, there are various modules; but we will be primarily using numpy and scipy. So, we import numpy as np. So, let me press shift enter ok. So, now, we are in a position to declare arrays using numpy. So, numpy arrays, we can declare as a =np.ndarray; so, n-dimensional array and then, we have to give certain inputs to this particular function. So, ndarray is a function inside np.

(Refer Slide Time: 15:55)



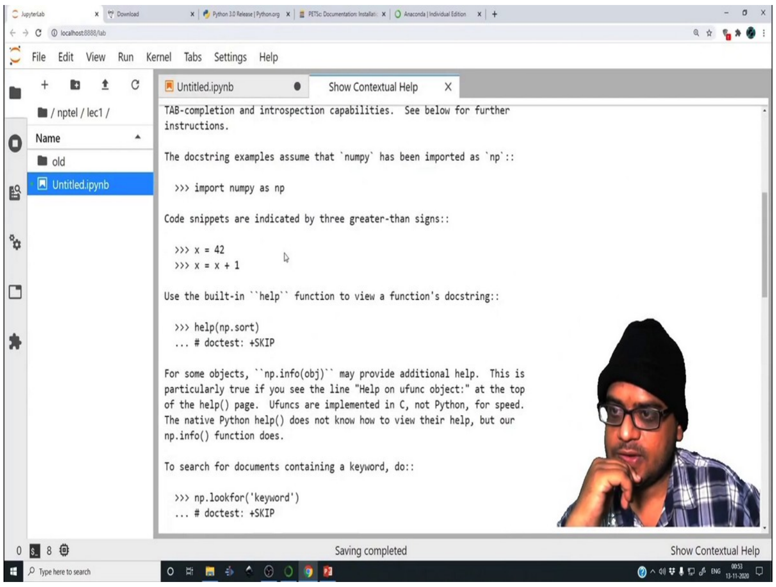
```
[8]: print(                    , " and b = ", b);
     The va                    1
[12]: import
[ ]: a = np
```

(Refer Slide Time: 16:01)



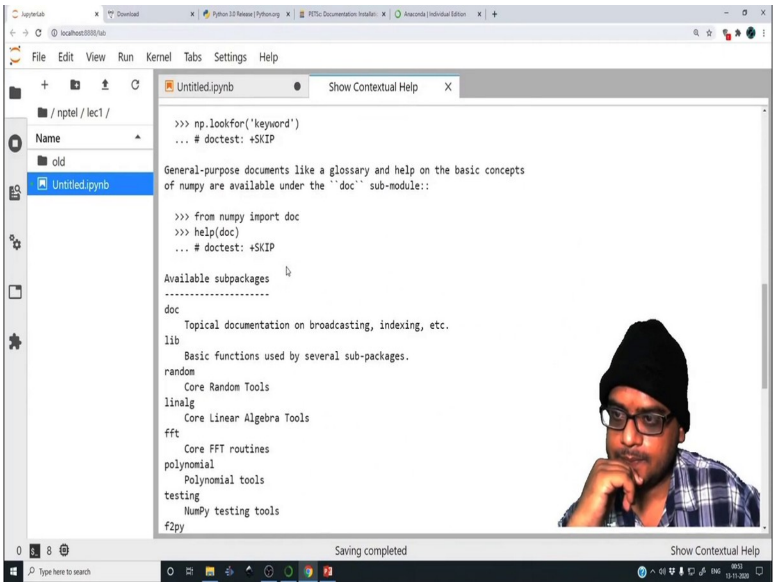
```
Type:        module
String form: <module 'numpy' from 'F:\\anaconda\\lib\\site-packages\\numpy\\__init__.py'>
File:        f:\anaconda\lib\site-packages\numpy\__init__.py
Docstring:
NumPy
=====

Provides
  1. An array object of arbitrary homogeneous items
  2. Fast mathematical operations over arrays
  3. Linear Algebra, Fourier Transforms, Random Number Generation

How to use the documentation
----------------------------
Documentation is available in two forms: docstrings provided
with the code, and a loose standing reference guide, available from
`the NumPy homepage <https://www.scipy.org&gt;`_.

We recommend exploring the docstrings using
`IPython <https://ipython.org&gt;`_, an advanced Python shell with
TAB-completion and introspection capabilities.  See below for further
instructions.

The docstring examples assume that `numpy` has been imported as `np`::

  >>> import numpy as np
```

(Refer Slide Time: 16:06)



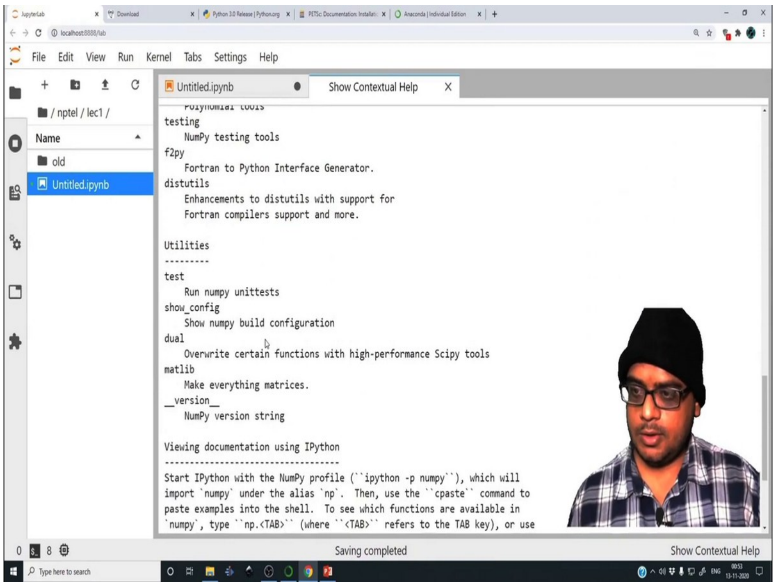
```
TAB-completion and introspection capabilities.  See below for further
instructions.

The docstring examples assume that `numpy` has been imported as `np`::

  >>> import numpy as np

Code snippets are indicated by three greater-than signs::

  >>> x = 42
  >>> x = x + 1

Use the built-in ``help`` function to view a function's docstring::

  >>> help(np.sort)
  ... # doctest: +SKIP

For some objects, ``np.info(obj)`` may provide additional help.  This is
particularly true if you see the line "Help on ufunc object:" at the top
of the help() page.  Ufuncs are implemented in C, not Python, for speed.
The native Python help() does not know how to view their help, but our
np.info() function does.

To search for documents containing a keyword, do::

  >>> np.lookfor('keyword')
  ... # doctest: +SKIP
```

(Refer Slide Time: 16:08)



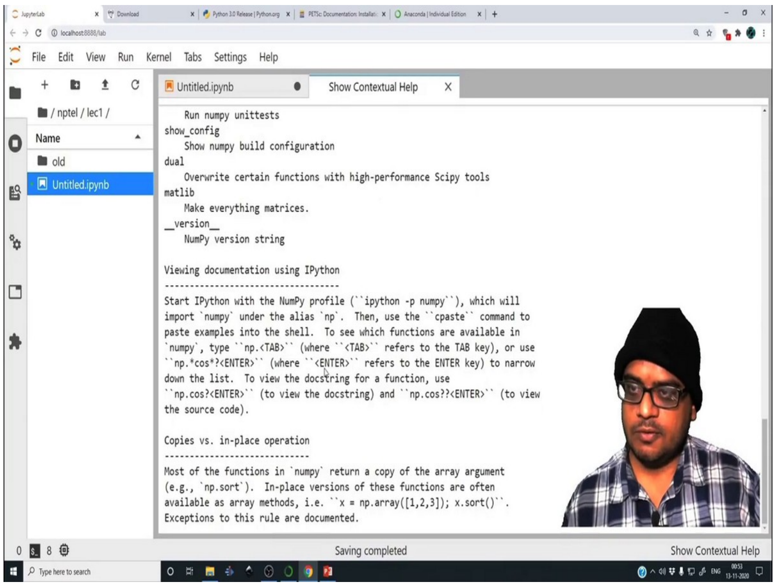
```
  >>> np.lookfor('keyword')
  ... # doctest: +SKIP

General-purpose documents like a glossary and help on the basic concepts
of numpy are available under the ``doc`` sub-module::

  >>> from numpy import doc
  >>> help(doc)
  ... # doctest: +SKIP

Available subpackages
---------------------
doc
    Topical documentation on broadcasting, indexing, etc.
lib
    Basic functions used by several sub-packages.
random
    Core Random Tools
linalg
    Core Linear Algebra Tools
fft
    Core FFT routines
polynomial
    Polynomial tools
testing
    NumPy testing tools
f2py
```

(Refer Slide Time: 16:18)



```
testing
    NumPy testing tools
f2py
    Fortran to Python Interface Generator.
distutils
    Enhancements to distutils with support for
    Fortran compilers support and more.

Utilities
---------
test
    Run numpy unittests
show_config
    Show numpy build configuration
dual
    Overwrite certain functions with high-performance Scipy tools
matlib
    Make everything matrices.
__version__
    NumPy version string

Viewing documentation using IPython
-----------------------------------
Start IPython with the NumPy profile (``ipython -p numpy``), which will
import `numpy` under the alias `np`.  Then, use the ``cpaste`` command to
paste examples into the shell.  To see which functions are available in
`numpy`, type ``np.<TAB>`` (where ``<TAB>`` refers to the TAB key), or use
```

(Refer Slide Time: 16:20)



```
    Run numpy unittests
show_config
    Show numpy build configuration
dual
    Overwrite certain functions with high-performance Scipy tools
matlib
    Make everything matrices.
__version__
    NumPy version string

Viewing documentation using IPython
-----------------------------------
Start IPython with the NumPy profile (``ipython -p numpy``), which will
import `numpy` under the alias `np`.  Then, use the ``cpaste`` command to
paste examples into the shell.  To see which functions are available in
`numpy`, type ``np.<TAB>`` (where ``<TAB>`` refers to the TAB key), or use
``np.*cos*?<ENTER>`` (where ``<ENTER>`` refers to the ENTER key) to narrow
down the list.  To view the docstring for a function, use
``np.cos?<ENTER>`` (to view the docstring) and ``np.cos??<ENTER>`` (to view
the source code).

Copies vs. in-place operation
-----------------------------
Most of the functions in `numpy` return a copy of the array argument
(e.g., `np.sort`).  In-place versions of these functions are often
available as array methods, i.e. ``x = np.array([1,2,3]); x.sort()``.
Exceptions to this rule are documented.
```

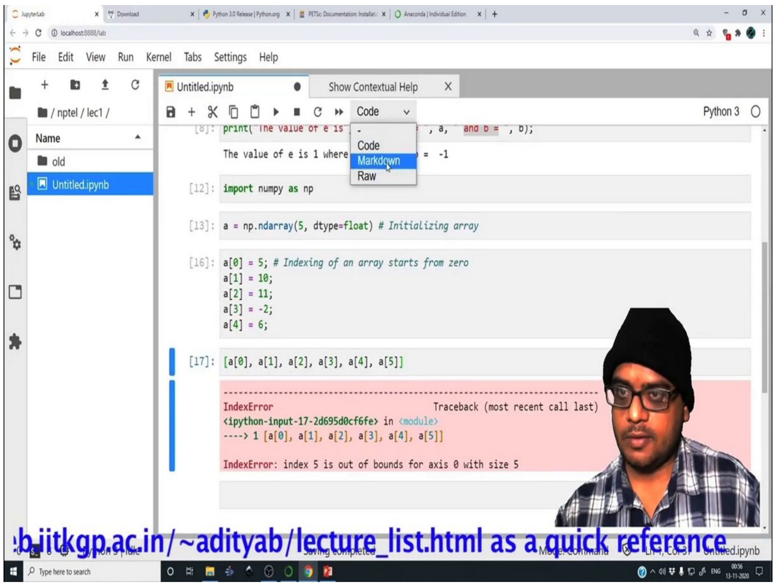
So, in case you want to know the reference for numpy, we can go to help, we can click on show contextual help, it will pop up this particular screen, we can double click on np and it will show the various functions that we have. So, we have random, we have linalg, we have fft, we have polynomial ok. So, we have a bunch of it will show a bunch of tools that exist and of course, numpy has a very good documentation online and you can always refer to the online documentation, if ever in doubt.

So, ndarray is a particular function that exists inside the module np. So, when we double click on ndarray, we will get the contextual help. So, ndarray shape, the data type ok, this is what we need to give. So, ndarray shape. So, let us give. So, let us make an array of size 5.

So, 5 dtype equal to float. So, let us print what a is. It is a random array. I mean it is something which is 0 and it is of size 5. So, 5 rows it has initialized. So, this is essentially a way of initializing arrays and data type float means whatever a will contain will be of type float ok.

So, in fact, let me write down the let me show the variable space. So, a is an ndarray, whose type is float 64. So, its 64 bit and it contains 5 elements. Now, how can I access the different elements ok. So, let me delete this cell. In order to delete the cell, we have to be out of focus from the cell and press d twice, it will delete the cell. Alternately, you can of course click on this particular cell, you can right click on it, you can delete cell ok.

(Refer Slide Time: 18:29)



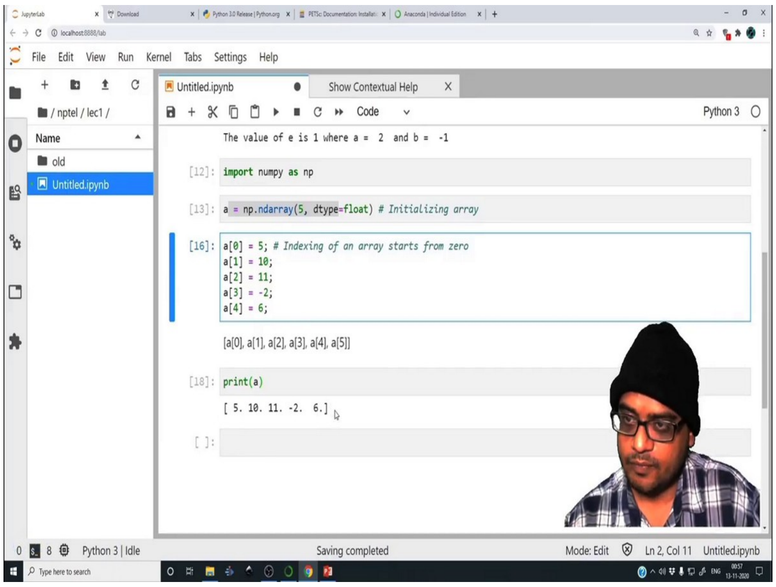
```
[8]: print("The value of e is" ... ", a, " and b = ", b);
     The value of e is 1 where ... = -1

[12]: import numpy as np

[13]: a = np.ndarray(5, dtype=float) # Initializing array

[16]: a[0] = 5; # Indexing of an array starts from zero
      a[1] = 10;
      a[2] = 11;
      a[3] = -2;
      a[4] = 6;

[17]: [a[0], a[1], a[2], a[3], a[4], a[5]]

---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-17-2d695d0cf6fe> in <module>
----> 1 [a[0], a[1], a[2], a[3], a[4], a[5]]

IndexError: index 5 is out of bounds for axis 0 with size 5
```

File Edit View Run Kernel Tabs Settings Help
Untitled.ipynb
Show Contextual Help
Code
Markdown
Raw
Python 3
/ nptel / lec1 /
old
Untitled.ipynb
...iitkgp.ac.in/~adityab/lecture_list.html as a quick reference

So, we can declare various elements of a. So, a[0], let me write it as 5; a[1], we can write it as 10; a[2], we can write it as 11; a[3], we can put it as -2 and a[4], we can write as 6. So, the indexing for an array starts from 0 and in case you have noticed this whatever text, I write after this hashtag is interpreted as a comment.

So, comment is something which is not executed along with the code ok. So, once I execute this, it would have assigned all the various indices of the array. So, an array; so, let me convert this cell into markdown. So, an array is something like this a[0], a[1], a[2], a[3], a[4], a[5] ok. This is how that; sorry, I have to convert into markdown ok.

(Refer Slide Time: 19:52)



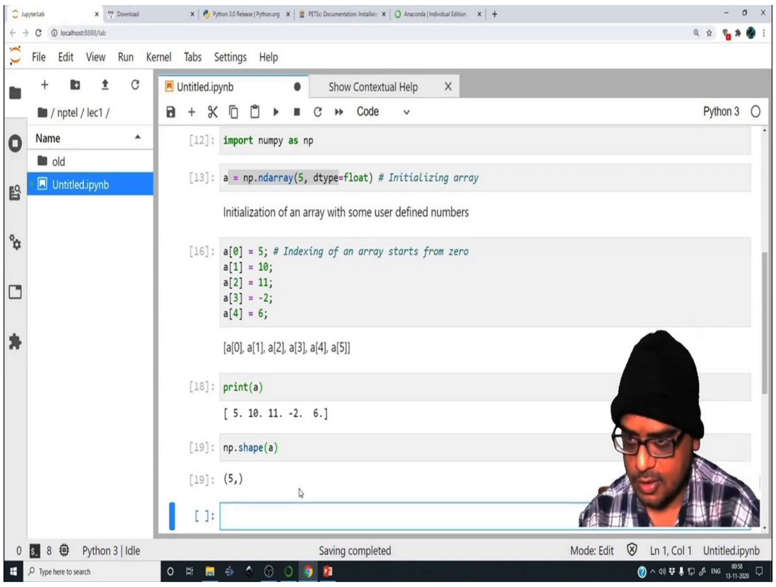
```
The value of e is 1 where a =  2  and b =  -1

[12]: import numpy as np

[13]: a = np.ndarray(5, dtype=float) # Initializing array

[16]: a[0] = 5; # Indexing of an array starts from zero
      a[1] = 10;
      a[2] = 11;
      a[3] = -2;
      a[4] = 6;
```

[a[0], a[1], a[2], a[3], a[4], a[5]]

```
[18]: print(a)
      [ 5. 10. 11. -2.  6.]

[ ]:
```

So, this is how that array looks. So, now, let us since we have declared the different elements of that array, let us see how that array looks. So, print a; oh, sorry. Yeah. So, now we have initialized the array over here. In this particular cell, we have declared the different elements of that array and then, we have printed out the elements of that array. Let me insert a cell above this. So, the way to insert a cell is to press a.

(Refer Slide Time: 20:37)



```
[12]: import numpy as np

[13]: a = np.ndarray(5, dtype=float) # Initializing array
```

Initialization of an array with some user defined numbers

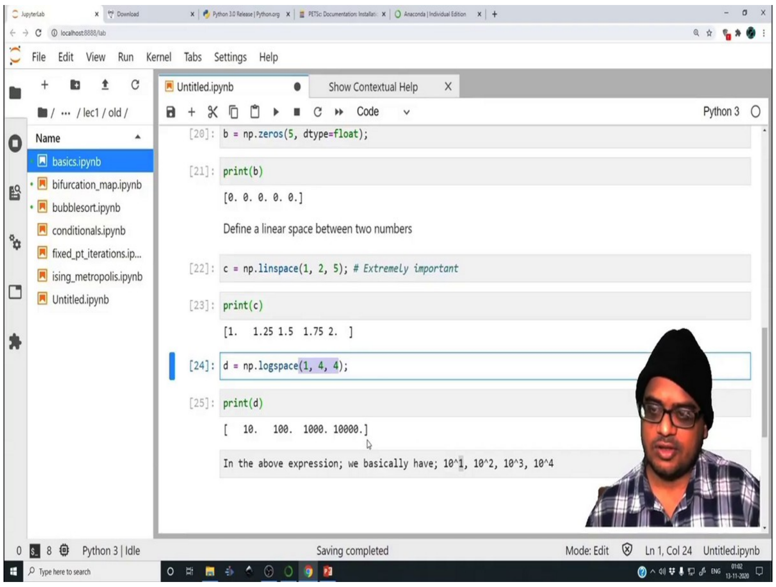
```
[16]: a[0] = 5; # Indexing of an array starts from zero
      a[1] = 10;
      a[2] = 11;
      a[3] = -2;
      a[4] = 6;
```

[a[0], a[1], a[2], a[3], a[4], a[5]]

```
[18]: print(a)
      [ 5. 10. 11. -2.  6.]

[19]: np.shape(a)

[19]: (5,)

[ ]:
```

So, let me convert this cell into markdown. So, we can write initialize of an array with some user defined numbers. So, this is how we declare an array. We can query the shape of the array. So, suppose you are given an array, I can query. So, np.shape(a.) If I print this, it will show (5,). So, its 5 rows rather it contains 5 elements. So, python does not differentiate much if you are declaring an array between rows and columns ok, that is good.

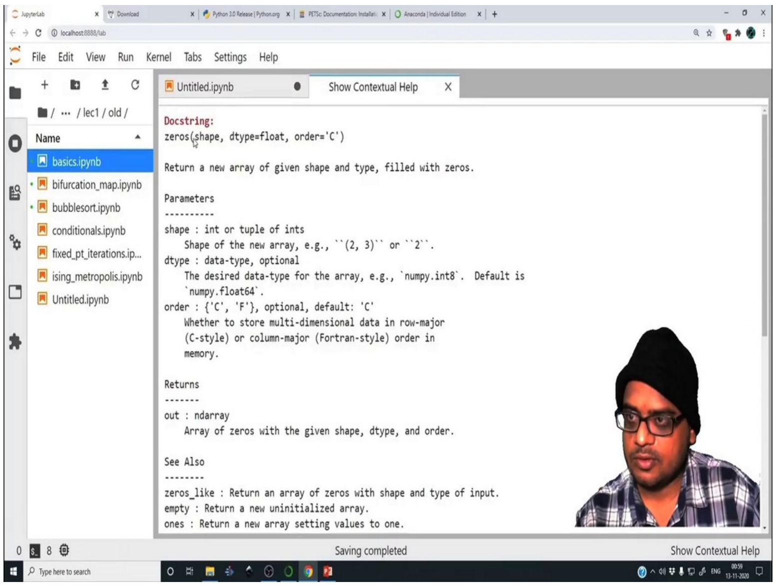
(Refer Slide Time: 21:40)



```
[20]: b = np.zeros(5, dtype=float);

[21]: print(b)

      [0. 0. 0. 0. 0.]

      Define a linear space between two numbers

[22]: c = np.linspace(1, 2, 5); # Extremely important

[23]: print(c)

      [1.   1.25 1.5  1.75 2.  ]

[24]: d = np.logspace(1, 4, 4);

[25]: print(d)

      [   10.   100.  1000. 10000.]

      In the above expression; we basically have; 10^1, 10^2, 10^3, 10^4
```

Now, let us come to some different kinds of arrays. So, we have initialized an array, but there are various kinds of initializations as well. So, let us initialize b as np.zeros(). Now, suppose you want to know the function reference for zeros.

(Refer Slide Time: 21:46)



```
Docstring:
zeros(shape, dtype=float, order='C')

Return a new array of given shape and type, filled with zeros.

Parameters
----------
shape : int or tuple of ints
    Shape of the new array, e.g., ``(2, 3)`` or ``2``.
dtype : data-type, optional
    The desired data-type for the array, e.g., `numpy.int8`.  Default is
    `numpy.float64`.
order : {'C', 'F'}, optional, default: 'C'
    Whether to store multi-dimensional data in row-major
    (C-style) or column-major (Fortran-style) order in
    memory.

Returns
-------
out : ndarray
    Array of zeros with the given shape, dtype, and order.

See Also
--------
zeros_like : Return an array of zeros with shape and type of input.
empty : Return a new uninitialized array.
ones : Return a new array setting values to one.
```

So, we just look. So, zeros shape and data type fine. So, zeros. So, let me define it as size 4 or let me make it 5 and dtype= float. So, now, let us print b; it is all zeros. Let me define c as a linear space. So, let me make this as a markdown. So, a linear space is defined as a as an array which is having elements which are linearly spaced. So, let us look at the syntax.

So, c = np.linspace(). So, linspace has the has a it takes input a, b and the number of points; where a is the first point, so let us say 1; b is the second point and how many points you want between 1 and 2? So, let me take 5 points. Let me print c. So, c is an array which goes from 1 to 2, if you look at this over here, it goes from one all the way to 2 and 5 is the number of elements in c.

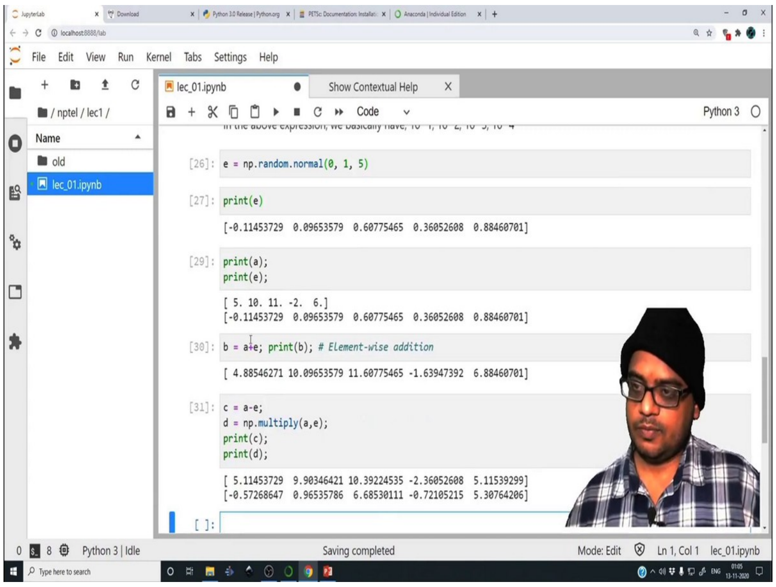
So, it has 1 2 3 4 5. So, this is how we can declare a linspace. It is a very useful command. It will be extremely useful to define meshes to define grids on which we will solve various kinds of equations. So, this is extremely important.

Another important function that is quite important which is quite relevant for this computational course is logspace. So, let us define d =np.logspace(1, 4,4) and let me take 4 elements ok. So, let us see what d is. So, now, logspace, it defines a linspace going from 1 to 4 and it makes 4 step. So, 1, 2, 3, 4; these are the 4 elements that you will create.

But now because it is a logspace, it will make 10 to the power first element, 10 to the power second element, 10 to the power third element. So, in the above expression, we basically have 10 to the power 1, 10 to the power 2, 10 to the power 3, 10 to the power 4 because these exponents are what we have declared over here inside the function.

So, this is a very easy way of generating a geometric progression of values ok. It is essentially a way to generate geometric progressions. Linspace is a way of generating arithmetic progression or logspace is for geometric progression.

(Refer Slide Time: 25:11)



```
[26]: e = np.random.normal(0, 1, 5)

[27]: print(e)

      [-0.11453729  0.09653579  0.60775465  0.36052608  0.88460701]

[29]: print(a);
      print(e);

      [ 5. 10. 11. -2.  6.]
      [-0.11453729  0.09653579  0.60775465  0.36052608  0.88460701]

[30]: b = a+e; print(b); # Element-wise addition

      [ 4.88546271 10.09653579 11.60775465 -1.63947392  6.88460701]

[31]: c = a-e;
      d = np.multiply(a,e);
      print(c);
      print(d);

      [ 5.11453729  9.90346421 10.39224535 -2.36052608  5.11539299]
      [-0.57268647  0.96535786  6.68530111 -0.72105215  5.30764206]

[ ]:
```

Now, let us add 2 arrays ok. So, this is a very useful thing to do. So, e=np. Random.normal (0,1,5) print e. So, do not worry about what I have written. It is essentially I am using the np module to use the random module to generate a normally distributed variable with average 0 and variance 1 and the number of elements is 5.

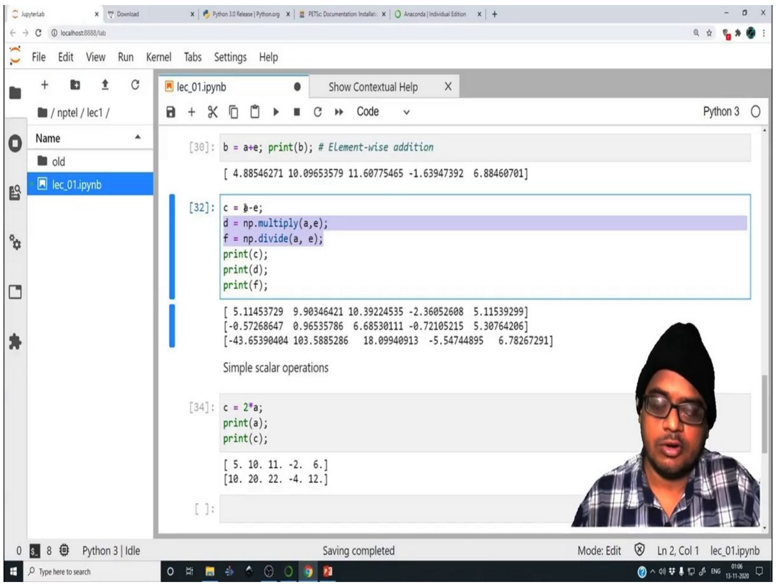
So, in case you are confused on the syntax, you can click on this and it shows ok. So, location, scale and size. So, we have generated e. So, let me just quickly print out two things; but first let us save this file, I think it got saved. Yeah, its saved over here. Let me rename this; so, lec 01.

So, let me print out some of the arrays that we can use to demonstrate ok. So, we have these two arrays; the first array is a, the second array is e. So, let us define b=a + e and let us print b as well. So, we have added this. So, it is an element wise addition ok. So, this is an addition.

So, it takes each element of a and e and so, a [0] is added with e[0], a[1] is added with e[1] and so on and it is assigned to elements of b. So, this is a very simple way of doing element wise addition. Similarly, we can easily do element wise subtraction, element wise multiplication.

But this is not recommended. It is not recommended to do element wise multiplication like this, it is better to do np.multiply(a,e). So, let us print out c. Let us print out d as well ok. So, this is how you obtain element wise subtraction. So, you can easily verify using these arrays and element wise multiplication as well.

(Refer Slide Time: 28:03)



```
[30]: b = a+e; print(b); # Element-wise addition

[ 4.88546271 10.09653579 11.60775465 -1.63947392  6.88460701]

[32]: c = a-e;
d = np.multiply(a,e);
f = np.divide(a, e);
print(c);
print(d);
print(f);

[ 5.11453729  9.90346421 10.39224535 -2.36052608  5.11539299]
[-0.57268647  0.96535786  6.68530111 -0.72105215  5.30764206]
[-43.65390404 103.5885286   18.09940913  -5.54744895   6.78267291]
```

Simple scalar operations

```
[34]: c = 2*a;
print(a);
print(c);

[ 5. 10. 11. -2.  6.]
[10. 20. 22. -4. 12.]
```

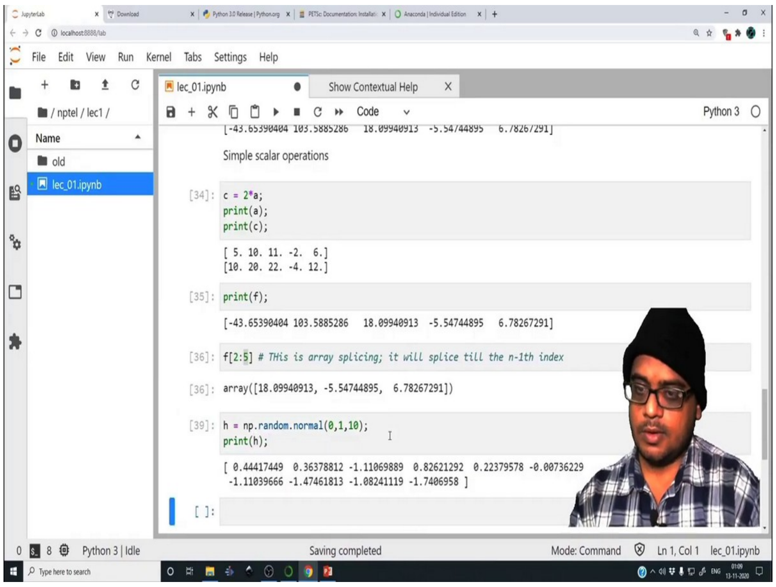
Similarly, we can do an element wise divide as well. So, the beauty about using cells is you can modify it and rerun the cell. Whatever is there in the previously run cells, it stays and that new cell that you are executing, all the variables inside that get updated ok. So, yeah. So, this is how you do element wise scalar operations.

You can obviously, add a single scalar to all this as well and it is much easier. You can simply do. So, let us make this simple scalar operations. So, I can make c= 2* a; let me print out what c is. In fact, let me print out what a was as well for reference.

So, with the help of this, we have multiplied all the elements of a with the scalar 2. So, scalar multiplication is quite straightforward. Just remember when you are doing element wise operation between two arrays in particularly multiplication and division, it is better to use np dot multiply and np dot divide.

In fact, we can use np.subtract and np.add also in order to avoid certain mishaps that can happen when you add elements ok. So, this is how we can perform simple operations using arrays. A very important aspect of arrays is to pick out various elements of an array.

(Refer Slide Time: 29:49)



```
[-43.65390404 103.5885286   18.09940913  -5.54744895   6.78267291]
```

Simple scalar operations

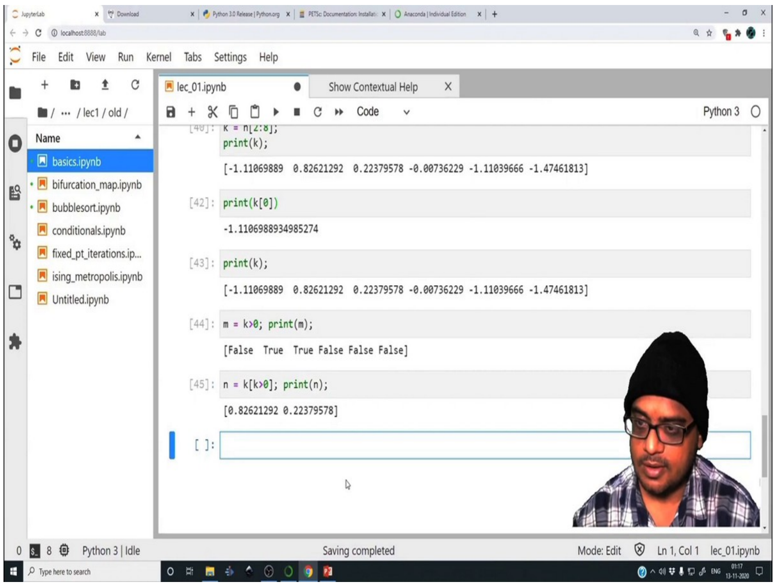
```
[34]: c = 2*a;
      print(a);
      print(c);

      [ 5. 10. 11. -2.  6.]
      [10. 20. 22. -4. 12.]

[35]: print(f);

      [-43.65390404 103.5885286   18.09940913  -5.54744895   6.78267291]

[36]: f[2:5] # THis is array splicing; it will splice till the n-1th index

[36]: array([18.09940913, -5.54744895,  6.78267291])

[39]: h = np.random.normal(0,1,10);
      print(h);

      [ 0.44417449  0.36378812 -1.11069889  0.82621292  0.22379578 -0.00736229
       -1.11039666 -1.47461813 -1.08241119 -1.7406958 ]

[ ]:
```

So, let us see, we have the array f. So, let me print f. Suppose, I want to pick out element number 2 to 4 of f. So, I will simply do f [2: 5]. So, this is called as array splicing. So, when I input 2, 5, it goes from; so, ok.

So, this is the; this is element number 0, element number 1, element number 2. So, this has chosen element number 2. So, 18 is what I have chosen from this. So, this is 0, 1, 2, 3, 4. So, the thing about array splicing is whatever index you put over here, it will return the n - 1th index.

So, if you put 5, it will return till element number 4. So, this is 0, 1, 2, 3, 4. So, this gives us elements 2, 3, 4. So, this is 2, 3 and 4 ok. Let me define a slightly longer array. Let me make h =np.random.normal(0, 1, 10) and let me print what the value of h is. So, now, suppose I want to choose the third element all the way to the nineth element; so, or rather to the eighth element.

(Refer Slide Time: 32:05)



```
[40]: k = n[2:8];
      print(k);

      [-1.11069889  0.82621292  0.22379578 -0.00736229 -1.11039666 -1.47461813]

[42]: print(k[0])

      -1.1106988934985274

[43]: print(k);

      [-1.11069889  0.82621292  0.22379578 -0.00736229 -1.11039666 -1.47461813]

[44]: m = k>0; print(m);

      [False  True  True False False False]

[45]: n = k[k>0]; print(n);

      [0.82621292 0.22379578]

[ ]:
```

So, I will do; so, I can splice it using h. So, the third element in this particular case is 2 because the numbering has an offset from 0. So, this is 0, 1, 2 ok. So, and I will go all the way to; so, this is 9, this is 8, this is 7. So, this is the index number 9. This is index number 0. This is the tenth element, but the index number is going to be 9.

So, I want it till the eighth element. So, I will simply put equal to 8 and let me assign this spliced array to something; let me assign it to k, then I can print out what k is. Excellent. So, we have chosen from the second index ok. So, this is 0, 1, 2 and we are going all the way till the seventh index.

So, 0, 1, 2, 3, 4, 5, 6, 7 ok. So, this is the last index. Another way of looking into it is the choice of the index number to the element number; but anyway, once you start using it, you will be comfortable in what it really means. You can have a look at the video as many times you want; but it should be comfortable to you, once you start using it. So, this is how you can splice an array and assign it to k.

Now, I can query the third element of k. So, the third element of k will be 2. So, I can print that particular element out as well. So, the third element is this. So, 0. So, this is 1, 2, 3; but the index number is 0, 1, 2 ok. Let me print out the first element of k that is index number 0.

So, it is this this particular value ok. So, the important thing to remember is arrays and matrices in fact have an offset from 0. This is very much similar to what happens in MATLAB as well. Rather, in C as well; in MATLAB in fact the offset is from 1.

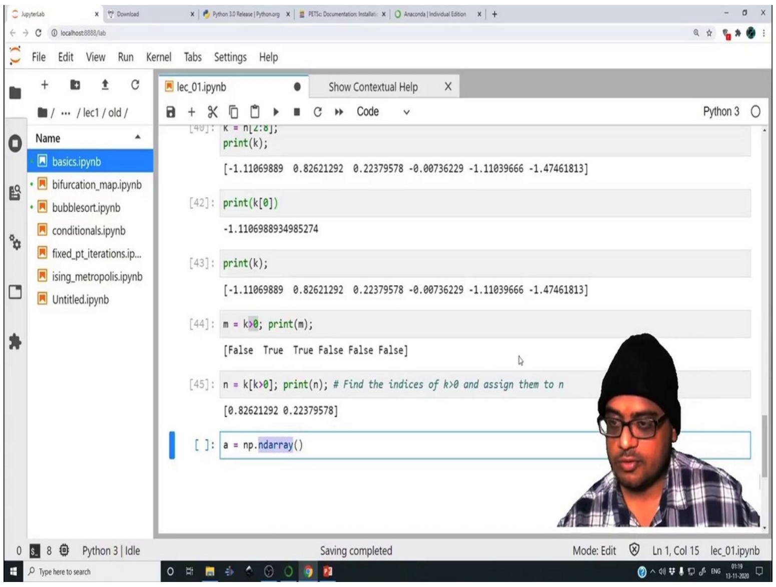
A similar set of a similar notebook will be made available in Octave as well. Whatever we are doing in Python, it will be made available in Octave; I will put up the download link to these notebooks. So, you can download them and run them on your own computer.

But in view of time, we are going to discuss some of the specifics and whatever you are able to do in Python, you should be able to do equivalently in Octave as well. So, with the help of the notebook files that you can download, you should be able to see how to do it. So, proceeding, let me show you a simple Booleans that you can do on various arrays. So, let me print again k for reference.

So, now, suppose I want to find out in the array k, whether there are elements which are greater than 0. So, let me just write m=k>0. Let me print m as well. So, m is an array which contains various Booleans or logicals. So, it says false.

So, k is not equal to 0, this particular element, so it says false, true, true, false, false, false. In fact, I can do this also. So, n = k[k >0]. So, it will let us see what it prints. So, let me print n. So, n only contains those elements of k which are greater than 0.

(Refer Slide Time: 36:12)

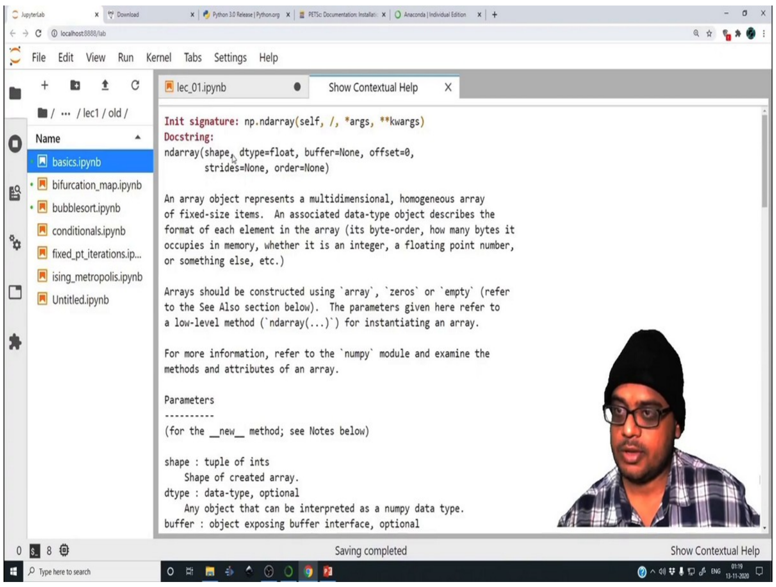


```
[40]: k = n[2:8];
      print(k);

      [-1.11069889  0.82621292  0.22379578 -0.00736229 -1.11039666 -1.47461813]

[42]: print(k[0])

      -1.1106988934985274

[43]: print(k);

      [-1.11069889  0.82621292  0.22379578 -0.00736229 -1.11039666 -1.47461813]

[44]: m = k>0; print(m);

      [False  True  True False False False]

[45]: n = k[k>0]; print(n); # Find the indices of k>0 and assign them to n

      [0.82621292 0.22379578]

[ ]: a = np.ndarray()
```

So, the meaning of this statement is finding out the indices of k which is greater than 0 and assign them to n or equivalently, I can write p equal to; so, yeah, I mean this is how you can obtain various conditionals without having to do a loop. So, traditionally, you would have to loop over each element of k and do a conditional whether k[i], if i is the iteration variable is greater than 0, it is less than 0; but now, you can do a scalar operation, you can broadcast, you can broadcast.

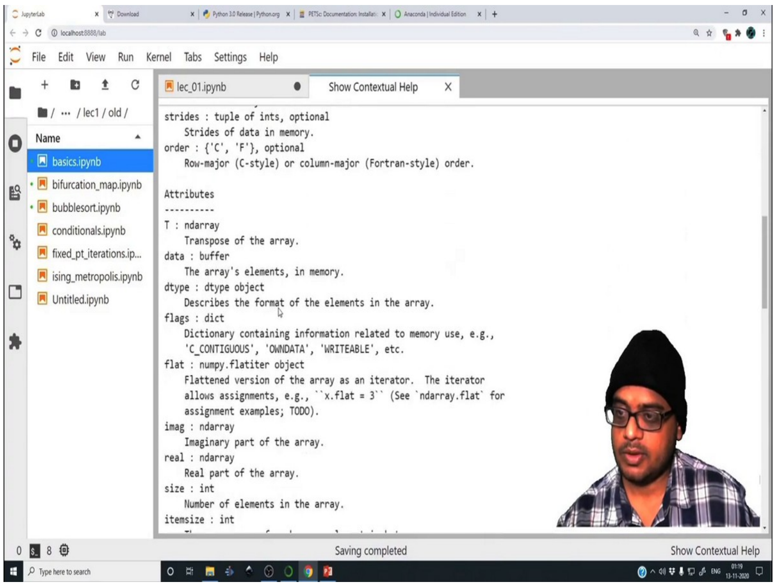
So, this particular check, you can apply on all the elements of k without having to worry about doing a loop over all the elements ok. So, we can declare array matrices as well. So, in the same way, we have declared the arrays, we can declare matrices as well. So, let me declare a =np.ndarray().

(Refer Slide Time: 37:37)



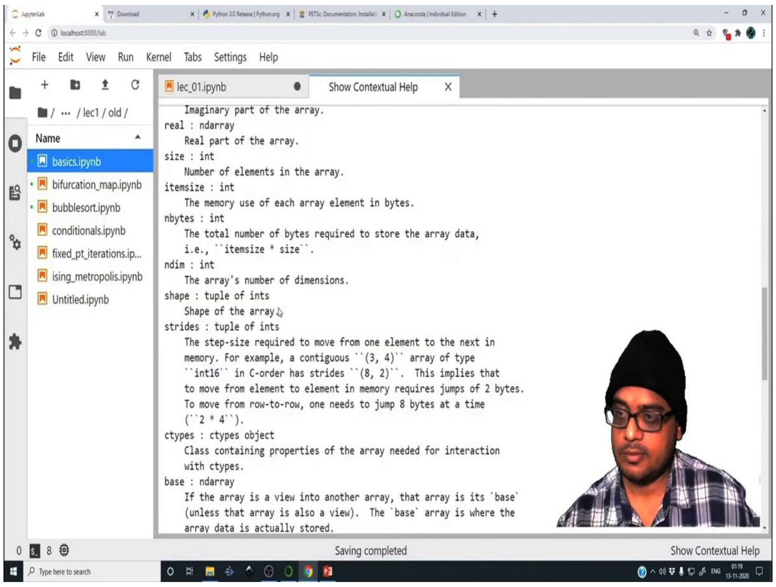
```
Init signature: np.ndarray(self, /, *args, **kwargs)
Docstring:
ndarray(shape, dtype=float, buffer=None, offset=0,
        strides=None, order=None)

An array object represents a multidimensional, homogeneous array
of fixed-size items.  An associated data-type object describes the
format of each element in the array (its byte-order, how many bytes it
occupies in memory, whether it is an integer, a floating point number,
or something else, etc.)

Arrays should be constructed using `array`, `zeros` or `empty` (refer
to the See Also section below).  The parameters given here refer to
a low-level method (`ndarray(...)`) for instantiating an array.

For more information, refer to the `numpy` module and examine the
methods and attributes of an array.

Parameters
----------
(for the __new__ method; see Notes below)

shape : tuple of ints
    Shape of created array.
dtype : data-type, optional
    Any object that can be interpreted as a numpy data type.
buffer : object exposing buffer interface, optional
```

(Refer Slide Time: 37:42)



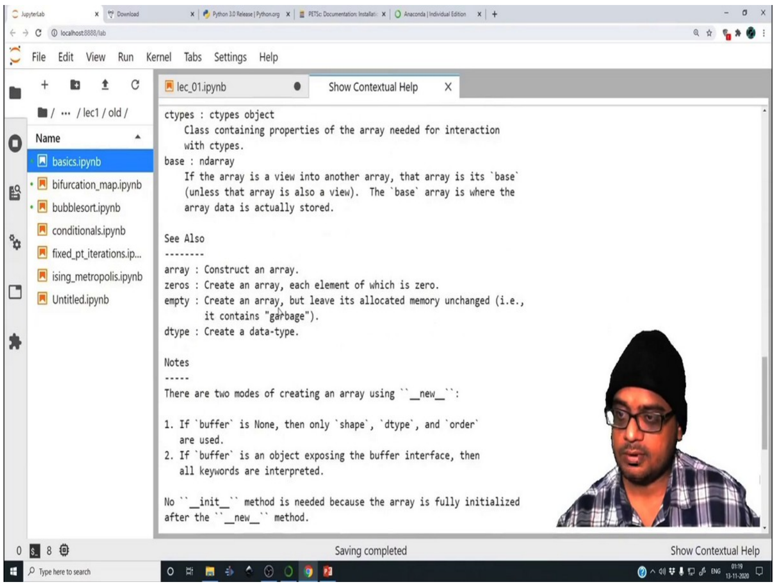
```
strides : tuple of ints, optional
    Strides of data in memory.
order : {'C', 'F'}, optional
    Row-major (C-style) or column-major (Fortran-style) order.

Attributes
----------
T : ndarray
    Transpose of the array.
data : buffer
    The array's elements, in memory.
dtype : dtype object
    Describes the format of the elements in the array.
flags : dict
    Dictionary containing information related to memory use, e.g.,
    'C_CONTIGUOUS', 'OWNDATA', 'WRITEABLE', etc.
flat : numpy.flatiter object
    Flattened version of the array as an iterator.  The iterator
    allows assignments, e.g., ``x.flat = 3`` (See `ndarray.flat` for
    assignment examples; TODO).
imag : ndarray
    Imaginary part of the array.
real : ndarray
    Real part of the array.
size : int
    Number of elements in the array.
itemsize : int
```

(Refer Slide Time: 37:43)



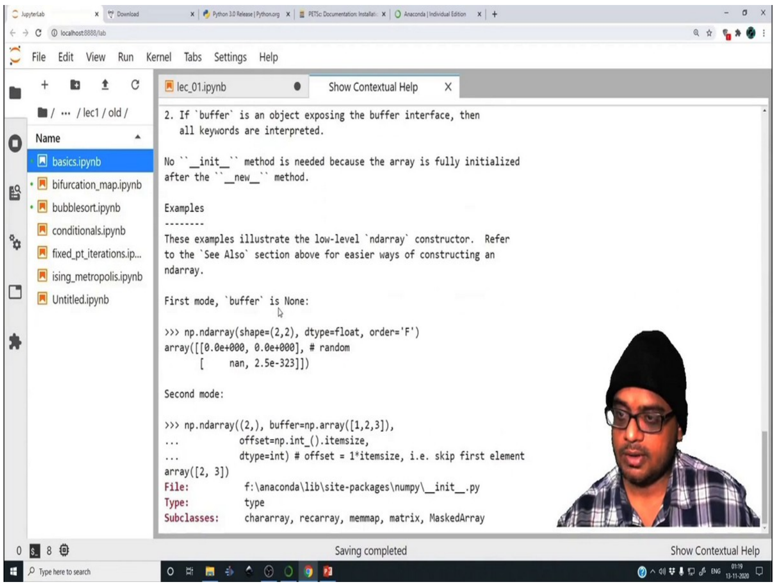
```
    Imaginary part of the array.
real : ndarray
    Real part of the array.
size : int
    Number of elements in the array.
itemsize : int
    The memory use of each array element in bytes.
nbytes : int
    The total number of bytes required to store the array data,
    i.e., ``itemsize * size``.
ndim : int
    The array's number of dimensions.
shape : tuple of ints
    Shape of the array.
strides : tuple of ints
    The step-size required to move from one element to the next in
    memory. For example, a contiguous ``(3, 4)`` array of type
    ``int16`` in C-order has strides ``(8, 2)``.  This implies that
    to move from element to element in memory requires jumps of 2 bytes.
    To move from row-to-row, one needs to jump 8 bytes at a time
    (``2 * 4``).
ctypes : ctypes object
    Class containing properties of the array needed for interaction
    with ctypes.
base : ndarray
    If the array is a view into another array, that array is its `base`
    (unless that array is also a view).  The `base` array is where the
    array data is actually stored.
```

(Refer Slide Time: 37:44)



```
ctypes : ctypes object
    Class containing properties of the array needed for interaction
    with ctypes.
base : ndarray
    If the array is a view into another array, that array is its `base`
    (unless that array is also a view).  The `base` array is where the
    array data is actually stored.

See Also
--------
array : Construct an array.
zeros : Create an array, each element of which is zero.
empty : Create an array, but leave its allocated memory unchanged (i.e.,
        it contains "garbage").
dtype : Create a data-type.

Notes
-----
There are two modes of creating an array using ``__new__``:

1. If `buffer` is None, then only `shape`, `dtype`, and `order`
   are used.
2. If `buffer` is an object exposing the buffer interface, then
   all keywords are interpreted.

No ``__init__`` method is needed because the array is fully initialized
after the ``__new__`` method.
```

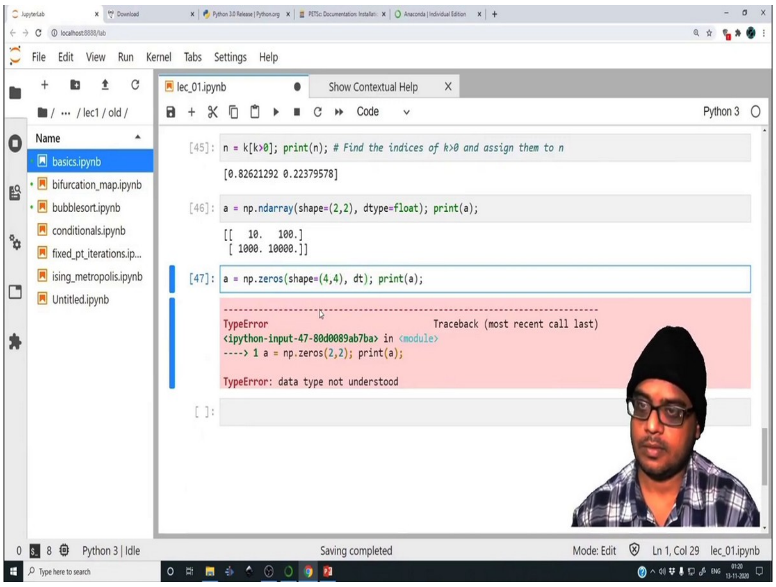
(Refer Slide Time: 37:45)



```
2. If `buffer` is an object exposing the buffer interface, then
   all keywords are interpreted.

No ``__init__`` method is needed because the array is fully initialized
after the ``__new__`` method.

Examples
--------
These examples illustrate the low-level `ndarray` constructor.  Refer
to the `See Also` section above for easier ways of constructing an
ndarray.

First mode, `buffer` is None:

>>> np.ndarray(shape=(2,2), dtype=float, order='F')
array([[0.0e+000, 0.0e+000], # random
       [     nan, 2.5e-323]])

Second mode:

>>> np.ndarray((2,), buffer=np.array([1,2,3]),
...            offset=np.int_().itemsize,
...            dtype=int) # offset = 1*itemsize, i.e. skip first element
array([2, 3])
File:           f:\anaconda\lib\site-packages\numpy\__init__.py
Type:           type
Subclasses:     chararray, recarray, memmap, matrix, MaskedArray
```

So, let us go to the documentation of ndarray quickly. So, it says shape and data type ok. So, we have to supply the shape and the data type.

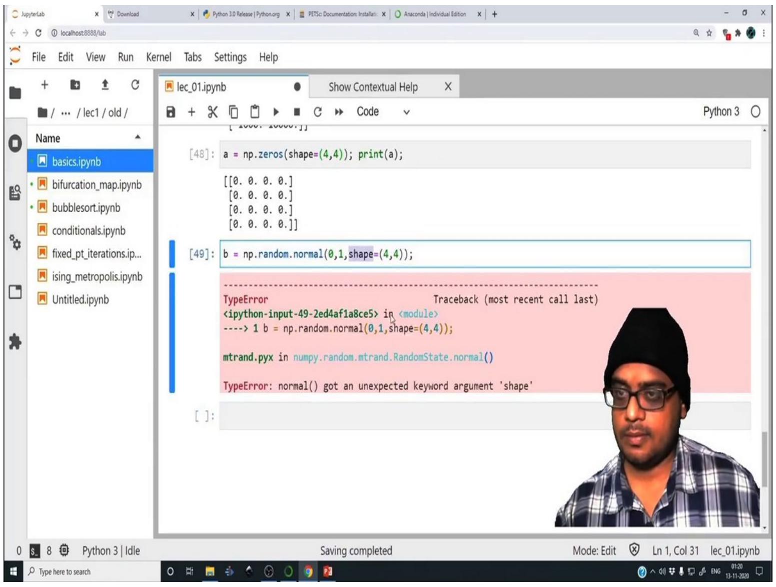
(Refer Slide Time: 37:55)



```
[45]: n = k[k>0]; print(n); # Find the indices of k>0 and assign them to n

[0.82621292 0.22379578]

[46]: a = np.ndarray(shape=(2,2), dtype=float); print(a);

[[   10.    100.]
 [ 1000. 10000.]]

[47]: a = np.zeros(shape=(4,4), dt); print(a);

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-47-80d0089ab7ba> in <module>
----> 1 a = np.zeros(2,2); print(a);

TypeError: data type not understood

[ ]:
```

It will be say 2 cross 2 and the data type will be float; let me print a. So, it has initialized this array to me, I mean whether a was declared as something like that.

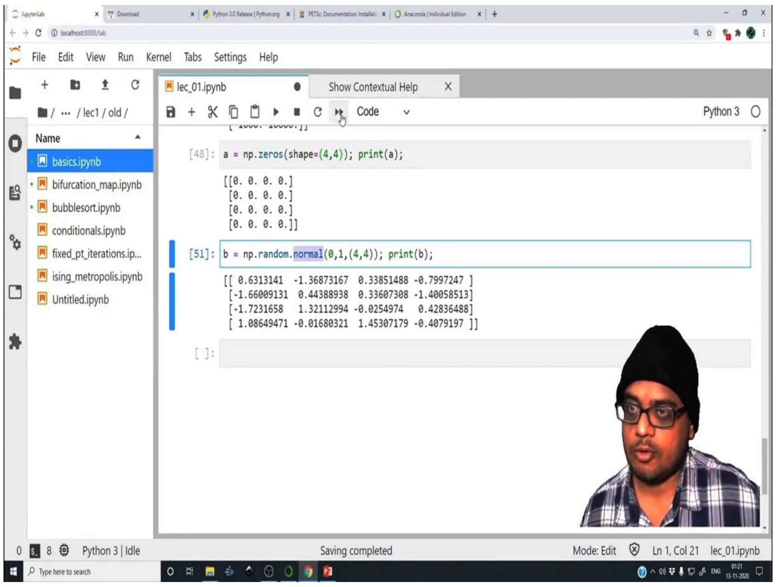
(Refer Slide Time: 38:31)



```
[48]: a = np.zeros(shape=(4,4)); print(a);

[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]

[49]: b = np.random.normal(0,1,shape=(4,4));

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-49-2ed4af1a8ce5> in <module>
----> 1 b = np.random.normal(0,1,shape=(4,4));

mtrand.pyx in numpy.random.mtrand.RandomState.normal()

TypeError: normal() got an unexpected keyword argument 'shape'

[ ]:
```

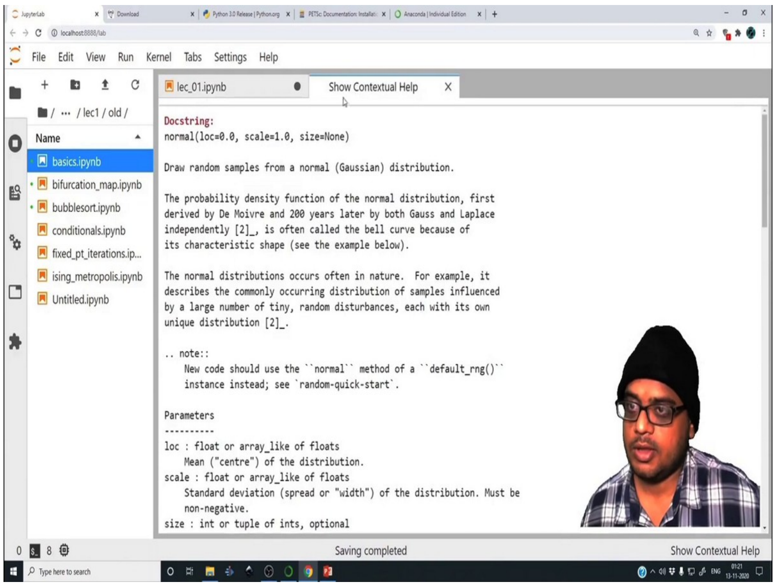
So, it initializes a 4 cross 4 matrix which is all zeros. In fact, we can make a random matrix as well. So, let me write np.random.normal(0,1,shape=[4,4]) ok.

(Refer Slide Time: 39:00)



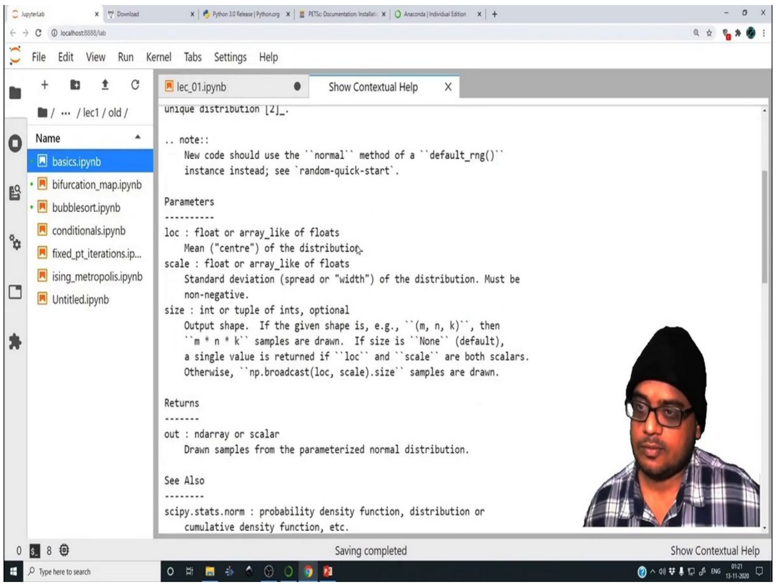
```
[48]: a = np.zeros(shape=(4,4)); print(a);

[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]

[51]: b = np.random.normal(0,1,(4,4)); print(b);

[[ 0.6313141  -1.36873167  0.33851488 -0.7997247 ]
 [-1.66009131  0.44388938  0.33607308 -1.40058513]
 [-1.7231658   1.32112994 -0.0254974   0.42836488]
 [ 1.08649471 -0.01680321  1.45307179 -0.4079197 ]]

[ ]:
```

(Refer Slide Time: 39:13)



```
Docstring:
normal(loc=0.0, scale=1.0, size=None)

Draw random samples from a normal (Gaussian) distribution.

The probability density function of the normal distribution, first
derived by De Moivre and 200 years later by both Gauss and Laplace
independently [2]_, is often called the bell curve because of
its characteristic shape (see the example below).

The normal distributions occurs often in nature.  For example, it
describes the commonly occurring distribution of samples influenced
by a large number of tiny, random disturbances, each with its own
unique distribution [2]_.

.. note::
    New code should use the ``normal`` method of a ``default_rng()``
    instance instead; see `random-quick-start`.

Parameters
----------
loc : float or array_like of floats
    Mean ("centre") of the distribution.
scale : float or array_like of floats
    Standard deviation (spread or "width") of the distribution. Must be
    non-negative.
size : int or tuple of ints, optional
```

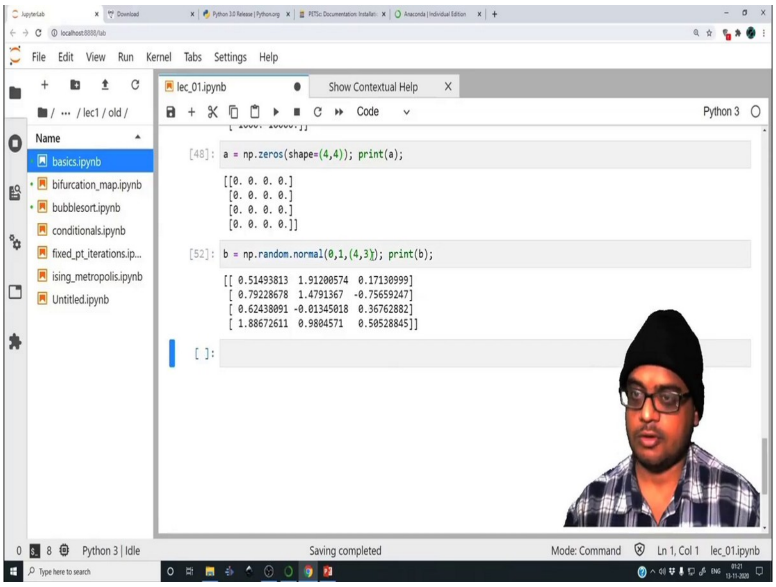
(Refer Slide Time: 39:18)



```
unique distribution [2]_.

.. note::
    New code should use the ``normal`` method of a ``default_rng()``
    instance instead; see `random-quick-start`.

Parameters
----------
loc : float or array_like of floats
    Mean ("centre") of the distribution.
scale : float or array_like of floats
    Standard deviation (spread or "width") of the distribution. Must be
    non-negative.
size : int or tuple of ints, optional
    Output shape.  If the given shape is, e.g., ``(m, n, k)``, then
    ``m * n * k`` samples are drawn.  If size is ``None`` (default),
    a single value is returned if ``loc`` and ``scale`` are both scalars.
    Otherwise, ``np.broadcast(loc, scale).size`` samples are drawn.

Returns
-------
out : ndarray or scalar
    Drawn samples from the parameterized normal distribution.

See Also
--------
scipy.stats.norm : probability density function, distribution or
    cumulative density function, etc.
```

So, this is how we supplied the size of the array over here. So, if you look into the contextual help for normal, you will see that you have to supply the size as well ok. So, the size is to be given as a tuple. So, this is called as a tuple.

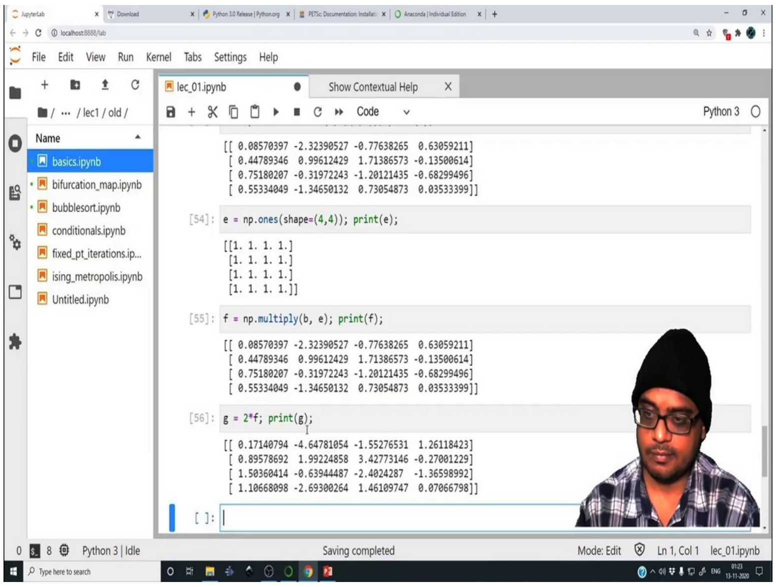
(Refer Slide Time: 39:31)



```
[48]: a = np.zeros(shape=(4,4)); print(a);

[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]

[52]: b = np.random.normal(0,1,(4,3)); print(b);

[[ 0.51493813  1.91200574  0.17130999]
 [ 0.79228678  1.4791367  -0.75659247]
 [ 0.62438091 -0.01345018  0.36762882]
 [ 1.88672611  0.9804571   0.50528845]]

[ ]:
```

Basically, it is a ordered number 4 cross 4; in fact, if I can make it 4 cross 3, it will make 4 rows and 3 columns ok. So, this is how you can create array, create matrices. So, we can apply the same operations. Let me make it 4 cross 4 and a, let me declare another array as e equal to np dot ones.

(Refer Slide Time: 39:54)



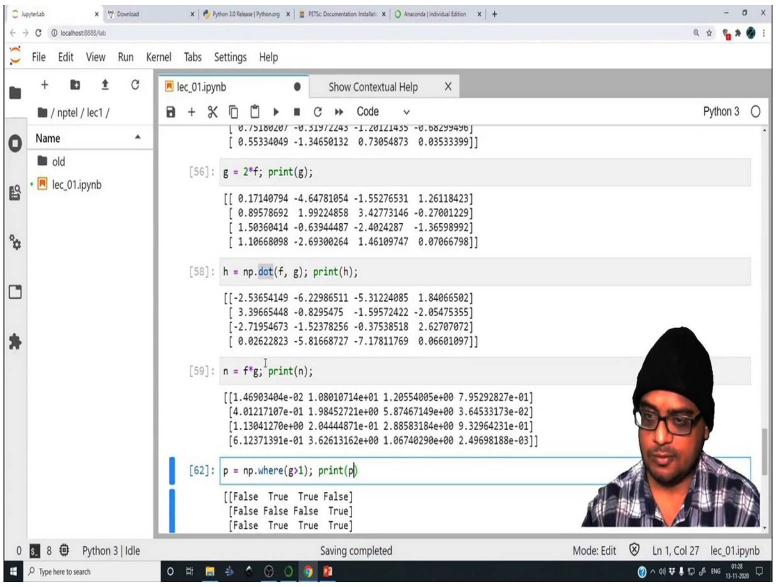
```
[[ 0.08570397 -2.32390527 -0.77638265  0.63059211]
 [ 0.44789346  0.99612429  1.71386573 -0.13500614]
 [ 0.75180207 -0.31972243 -1.20121435 -0.68299496]
 [ 0.55334049 -1.34650132  0.73054873  0.03533399]]

[54]: e = np.ones(shape=(4,4)); print(e);

[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]

[55]: f = np.multiply(b, e); print(f);

[[ 0.08570397 -2.32390527 -0.77638265  0.63059211]
 [ 0.44789346  0.99612429  1.71386573 -0.13500614]
 [ 0.75180207 -0.31972243 -1.20121435 -0.68299496]
 [ 0.55334049 -1.34650132  0.73054873  0.03533399]]

[56]: g = 2*f; print(g);

[[ 0.17140794 -4.64781054 -1.55276531  1.26118423]
 [ 0.89578692  1.99224858  3.42773146 -0.27001229]
 [ 1.50360414 -0.63944487 -2.4024287  -1.36598992]
 [ 1.10668098 -2.69300264  1.46109747  0.07066798]]

[ ]:
```

So, let us look into the documentation of ones; shape data type. So, the shape will be equal to 4 cross 4 and that is it. I mean we do not need anything else; we can print e. So, it creates an array in which all the elements are 1. So, now, we can do a matrix multiplication.

So, we can do np.multiply(b,e) ok. So, let us see what f is. So, quite naturally, all the element wise operations. So, b=f ok. So, similarly let me in fact, declare g=2*f. Let us print what g is ok. So, I have multiplied all the elements of the matrix f by 2.

So, now we can multiply two matrices in the sense of a true matrix multiplication. So, true matrix multiplication is not an element wise multiplication. So, if I were to multiply this matrix, so this entire matrix and this entire matrix, the first element of that matrix would be this row multiplied by this column; the second element would be this row multiplied by this column.

(Refer Slide Time: 41:55)



```
 [ 0.55334049 -1.34650132  0.73054873  0.03533399]]

[56]: g = 2*f; print(g);

[[ 0.17140794 -4.64781054 -1.55276531  1.26118423]
 [ 0.89578692  1.99224858  3.42773146 -0.27001229]
 [ 1.50360414 -0.63944487 -2.4024287  -1.36598992]
 [ 1.10668098 -2.69300264  1.46109747  0.07066798]]

[58]: h = np.dot(f, g); print(h);

[[-2.53654149 -6.22986511 -5.31224085  1.84066502]
 [ 3.39665448 -0.8295475  -1.59572422 -2.05475355]
 [-2.71954673 -1.52378256 -0.37538518  2.62707072]
 [ 0.02622823 -5.81668727 -7.17811769  0.06601097]]

[59]: n = f*g; print(n);

[[1.46903404e-02 1.08010714e+01 1.20554005e+00 7.95292827e-01]
 [4.01217107e-01 1.98452721e+00 5.87467149e+00 3.64533173e-02]
 [1.13041270e+00 2.04444871e-01 2.88583184e+00 9.32964231e-01]
 [6.12371391e-01 3.62613162e+00 1.06740290e+00 2.49698188e-03]]

[62]: p = np.where(g>1); print(p)

[[False  True  True False]
 [False False False  True]
 [False  True  True  True]
```

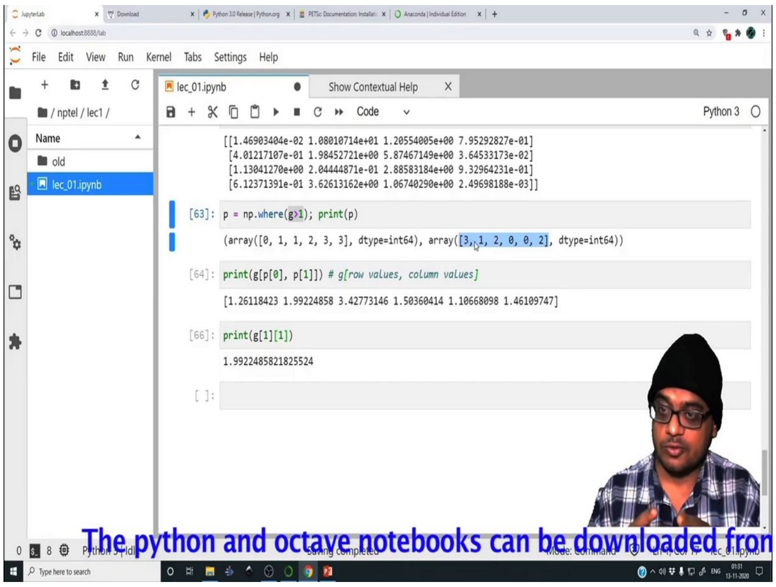
So, that is a true matrix multiplication and the way to do it. So, h = np.dot(f,g). In fact, let me print out what h is. In fact, one should always use these kind of functions to avoid any ambiguity; you may imagine writing f times g, you can simply write n=f*g.

But what is this multiplication? Is it an element wise multiplication between 2 numbers or is it an actual matrix multiplication in the sense that we know from class 11, 12? A matrix multiplication is not a element wise multiplication, it involves rows multiplying various columns ok. So, it is not the same; it is not the same ok.

So, let us do the Boolean. So, let us do the Boolean of; so, let us choose this array g, this matrix g ok. So, first of all we can choose various indices. So, we can choose the first element, let me print this out; print g and it is this first element. In fact, I can print out the first row, second column; it comes out to be this. So, this is how you can pick out the different elements of a matrix. Alternately, you can do the Boolean as well. So, let me print g <0 0; so, it says false, true.

So, this is less than 0; hence, this is true and so on. So, you can do all those things as well. Let me show you something useful as well. So, let us declare p =np.where(g>1); so, where is a kind of logical broadcasting. So, let us say where g greater than 1 suppose ok. So, let me print out p as well.

(Refer Slide Time: 44:22)



```
[[1.46903404e-02 1.08010714e+01 1.20554005e+00 7.95292827e-01]
 [4.01217107e-01 1.98452721e+00 5.87467149e+00 3.64533173e-02]
 [1.13041270e+00 2.04444871e-01 2.88583184e+00 9.32964231e-01]
 [6.12371391e-01 3.62613162e+00 1.06740290e+00 2.49698188e-03]]

[63]: p = np.where(g>1); print(p)

(array([0, 1, 1, 2, 3, 3], dtype=int64), array([3, 1, 2, 0, 0, 2], dtype=int64))

[64]: print(g[p[0], p[1]]) # g[row values, column values]

[1.26118423 1.99224858 3.42773146 1.50360414 1.10668098 1.46109747]

[66]: print(g[1][1])

1.9922485821825524

[ ]:
```

So, when I do p =np.where(g>1), this particular function call will give me all the indices of the matrix g, where this particular conditional is being satisfied. So, where is g greater than 1 being satisfied? At row number 0, column number 3.

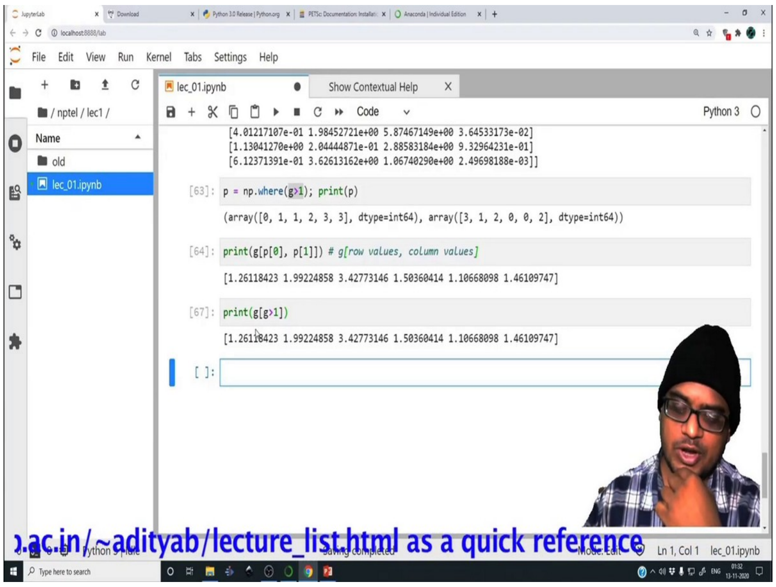
So, row number 0 is this and column number 3 is this. Remember this is column 0, column 1, column 2, column 3. Similarly, its being satisfied at row equal to 1, column equal to 1; this is row equal to 1 and this particular thing is column equal to 1 ok; similarly row equal to 1, column equal to 2; row 1, column 2.

So, as you can see np. where() is a very versatile function, where it will broadcast that conditional all over the array g or over the matrix g. It also works for arrays as well and it will tell you all the indices, where that particular conditional is true. So, in this particular case, it is true for all those pairs of i and j’s if you mean. So, then how can I get all the values where this conditional is satisfied? I want to. So, I have obtained all the indices. So, I can simply say let me print out g[p[0],p[1]).

So, when I do g [p[0], p[1]] this essentially means g of all those row values comma column values. So, g[p[0], p[1]] means g[0,3]. So, let me print g[0, 3]. So, it comes out to be this first value, then what is it g[1][1]? So, we get the second value.

So, instead of manually writing down each value, we can simply supply whatever indices we have find found out. So, these are the row indices and these are the column indices which satisfy that particular conditional. We will supply those row and column indices into the g to get back the values, where this particular conditional has been satisfied.

(Refer Slide Time: 47:28)


File Edit View Run Kernel Tabs Settings Help
lec_01.ipynb
Show Contextual Help
Python 3
/ nptel / lec1 /
Name
old
lec_01.ipynb
.ac.in/~adityab/lecture_list.html as a quick reference
Ln 1, Col 1
lec_01.ipynb

```
 [4.01217107e-01 1.98452721e+00 5.87467149e+00 3.64533173e-02]
 [1.13041270e+00 2.04444871e-01 2.88583184e+00 9.32964231e-01]
 [6.12371391e-01 3.62613162e+00 1.06740290e+00 2.49698188e-03]]

[63]: p = np.where(g>1); print(p)

(array([0, 1, 1, 2, 3, 3], dtype=int64), array([3, 1, 2, 0, 0, 2], dtype=int64))

[64]: print(g[p[0], p[1]]) # g[row values, column values]

[1.26118423 1.99224858 3.42773146 1.50360414 1.10668098 1.46109747]

[67]: print(g[g>1])

[1.26118423 1.99224858 3.42773146 1.50360414 1.10668098 1.46109747]

[ ]:
```

An alternate way of doing it, let me delete this which is simply g[g>1]. It gives us the same value. So, there are multiple ways of getting things done in Python. I have shown you how to declare scalars, how to declare arrays, how to declare matrices, how to do some simple operations on them. As the course progresses, we will look into the various functions that will find useful and we can have a small discussion about that particular function at that point of time.

I do not want to overwhelm you with so many functions. Many of you might be knowing Python, many of you might be new to Python or in fact, programming this is a very good way of starting off programming. Have a look at this. Try to implement these on your own.

You may run into some issues; you can e-mail me. You can Google, Google is a very good resource for learning Python as well. So, with this, we end this particular lecture. In the next class, we look at some basic programming constructs and a very simple program which makes use of those constructs.

Its bye from me. Until then, have a good day. Bye.