



IIT Madras

ONLINE DEGREE

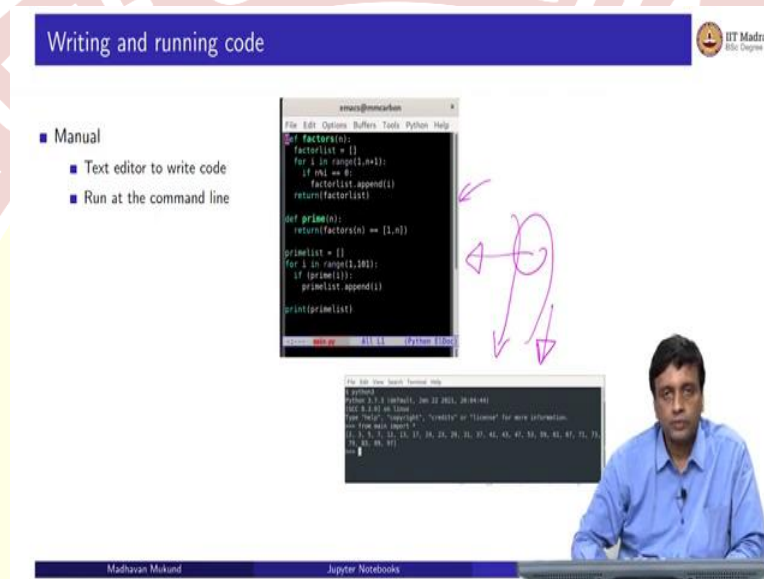
Programming Data Structures and Algorithms using Python

Professor Madhavan Mukund

Jupyter Notebooks

So, welcome to the course Programming Data Structures and Algorithms using Python. So, to begin with we will look at some of the things that you are already familiar with just to get a refresher on Python and one of the things I will start with is the environment in which we are going to run Python code. So, I am going to be talking to you today about Jupyter notebooks.

(Refer Slide Time: 0:32)



So, normally when we want to write and run code there are many options available to us, perhaps the simplest option is what you see here, so you have a text editor, so you type your code and then you open a kind of a console or a terminal and then you load the code that you have typed in in your text editor and then you run it.

Now, this is of course, easy to do but it is a little bit tedious because if you decide to make a change over here then you have to go back, reload it, so going around in this kind of cycle of edit and run is a little bit cumbersome if you are using a separate text editor and using directly the console.

(Refer Slide Time: 1:16)

Writing and running code

- Manual
 - Text editor to write code
 - Run at the command line
- Integrated Development Environment (IDE)
 - Single application to write and run code
 - On desktop or online, [replit](#)
 - Quick update-run cycle
 - Debugging, testing, ...

```
def factors(n):  
    factorlist = []  
    for i in range(1, n+1):  
        if n%i == 0:  
            factorlist.append(i)  
    return factorlist  
  
def primes(n):  
    return factors(n) == [1, n]  
  
primeslist = []  
for i in range(1, 101):  
    if primes(i):  
        primeslist.append(i)  
print(primeslist)
```

Output:
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 67, 71, 73, 79, 83, 89, 97]

So, this gives rise to a more convenient interface, which you are familiar with REPLIT called an Integrated Development Environment or IDE. So, in an IDE you have on one side something where you can edit your code and side by side you have this command window where you can run the code and see the output.

And because you can now see the output and edit your code in the same interface it is much easier to cycle back and forth and see how your code can be changed and how the output changes. So, you have this quick cycle of updating your code and running it, and of course, an IDE has other features also, so it offers you typically a debugger.

A debugger will allow you to interrupt your code and inspect the values of various variables to see what is going on. You might also be able to prepare some test cases in advance and run them to see how your code performs and so on. So, this is the default that most people use for developing serious amounts of code, and of course different IDEs are designed for different programming languages.

Some like REPLIT support multiple languages; other IDEs you might find are dedicated to certain languages like Python or Java or C plus plus and so on. So, we already have an IDE but I am going to present to you yet another way of writing and running code and the question is why would one want one more when you already have an IDE.

(Refer Slide Time: 2:48)

Collaboration



- Share your code
 - Collaborative development
 - Report your results

```
jupyter PDSA Week 1 Lecture 1 [username]
File Edit View Insert Cell Kernel Widgets Help Home Python 3.0

Compute primes from 1 to 100

In [1]: def factors(n):
        factorsList = []
        for i in range(1, n+1):
            if n%i == 0:
                factorsList.append(i)
        return(factorsList)

    def prime(n):
        return(factors(n) == [1, n])

    primesList = []
    for i in range(1, 101):
        if prime(i):
            primesList.append(i)
    print(primesList)

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



Madhavan Mukund

Jupyter Notebooks

Collaboration



- Share your code
 - Collaborative development
 - Report your results
- Documentation
 - Interleave with the code

```
jupyter PDSA Week 1 Lecture 1 [username]
File Edit View Insert Cell Kernel Widgets Help Home Python 3.0

Compute primes from 1 to 100

In [1]: def factors(n):
        factorsList = []
        for i in range(1, n+1):
            if n%i == 0:
                factorsList.append(i)
        return(factorsList)

    def prime(n):
        return(factors(n) == [1, n])

    primesList = []
    for i in range(1, 101):
        if prime(i):
            primesList.append(i)
    print(primesList)

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

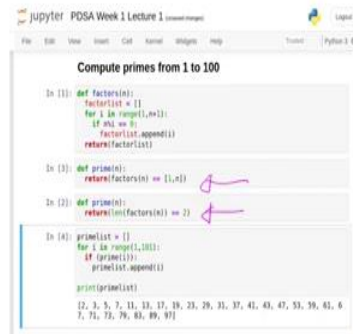


Madhavan Mukund

Jupyter Notebooks

सिद्धिर्भवति कर्मजा

- Share your code
 - Collaborative development
 - Report your results
- Documentation
 - Interleave with the code
- Switch between different versions of code



```

In [1]: def factors(n):
        factorlist = []
        for i in range(1, n+1):
            if n%i == 0:
                factorlist.append(i)
        return factorlist

In [2]: def prime(n):
        return len(factors(n)) == 2

In [3]: def primes():
        primelist = []
        for i in range(1, 101):
            if prime(i):
                primelist.append(i)
        print(primelist)

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

```



So the main point that I would like to emphasize is that writing code whether for yourself or for teaching, like I am teaching now, also involves collaboration, very rarely do you write code only for yourself, you usually work in a team. So, as a team you might be together developing the code or this might be a larger team.

For instance, it could be some kind of a research environment and this is very common, say in machine learning, you are trying to solve a problem, coding is part of the solution but is not the solution in itself. So, when you run the code not only it is possible that more than one person may be running the code or developing the code to run.

But there are maybe people who are not running the code, who are not even aware of how the code works but who might want to know what the code does. So, then you want to be able to share your results. Now, not notice that when you have a typical IDE, when you run the code you can see the results, but what happens after you run the code. You cannot take the code, you cannot take the IDE open to another office and show somebody what you have done.

You would like to have some way of preserving the output for later, without having to go through a sort of cumbersome process of saving it to a file or preparing a report or something like that. So, here on the right you see a typical view of what a Jupyter notebook, which is an interface that I am going to introduce today looks like.

So, you have on one side the ability to write text, so in this case is just a simple title, but you can also, as we will see, introduce more annotations which describe what you have done or what you are planning to do and then you have in this block, you have the actual code exactly

as you would have typed it into a standard editor or into an IDE and now at the bottom you have outside this grey area, you have the output of the code what you would normally see in a separate window on the side in an IDE.

So, the documentation is interleaved with the code, so you have the documentation here, you can have more documentation, so in between your code fragments or different parts of your code, you can introduce documentation without looking at it in the comment section of a Python program. So, in python of course, you can include comments by writing this hash and then writing something after that.

But this does not look very neat and it is also very difficult to format. The idea of this notebook is that you can actually format your text and interleave it with the code so that it looks readable. The other thing that you can do in this environment which is difficult to do when you are working in an IDE is to replace one piece of code by another piece of code without abandoning the previous one.

You do not want to throw it away, you want to say what if I replace this by that, so I might have two different definitions of the same function and I may want to try out both of them without having to go through a tedious process of loading a different file or deleting one and replacing it by the other, so I want both but only one to be active and we will see that in a notebook this is possible because it really depends on which was the last copy of that function that you executed.

(Refer Slide Time: 6:04)

The slide is titled "Collaboration" and features a list of capabilities on the left and a Jupyter Notebook interface on the right. The Jupyter Notebook shows a code cell with Python code for plotting a sine wave and its square, with the resulting plot displayed below the code.

Collaboration

- Share your code
 - Collaborative development
 - Report your results
- Documentation
 - Interleave with the code
- Switch between different versions of code
- Export and import your project
- Preserve your output

Jupyter Notebook Interface:

```
In [1]: import matplotlib.pyplot as plt
import numpy as np

# Data for plotting
t = np.arange(0.0, 2.0, 0.01)
s = 1 + np.sin(2 * np.pi * t)

fig, ax = plt.subplots()
ax.plot(t, s)

plt.show()
```

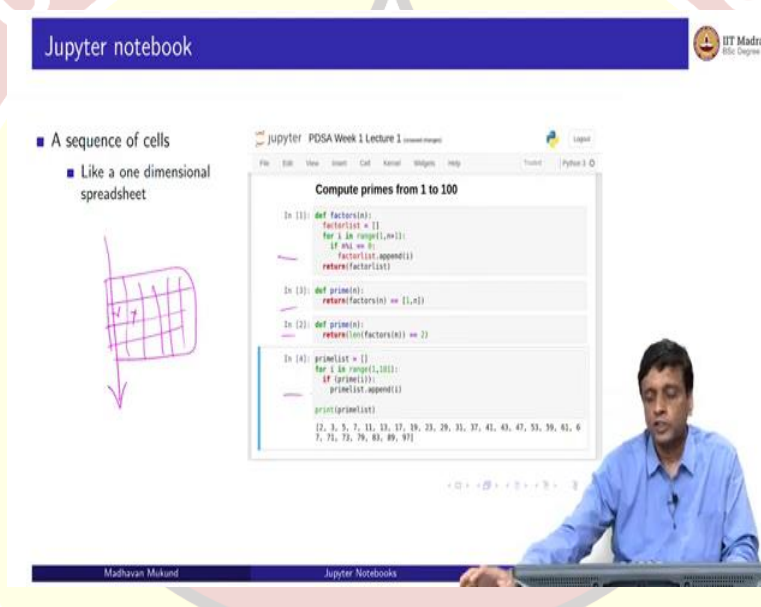
The plot shows a sine wave oscillating between 0 and 2 over the time interval [0, 2].

Madhavan Mukund | Jupyter Notebooks

And finally, as I said you would like to keep the outputs available for somebody else to see. Now, this could be for two reasons, one is you are just reporting it, you want to prepare a kind of documented output after you have run your code so that you can evaluate whether it did the job that it was supposed to do or you want to run it again.

And the other thing is also that if somebody else wants to run your code they should be able to see what you had got with your output, so that when they run it they can see whether they get the same output or they get some other output. So, preserving the output is part of saving this project as it were for future use.

(Refer Slide Time: 6:41)



The slide shows a Jupyter notebook titled "Jupyter notebook" with a "UT Madras" logo. It features a list of bullet points on the left: "A sequence of cells" and "Like a one dimensional spreadsheet". A hand-drawn diagram of a grid with arrows illustrates the sequence. The main area displays a Jupyter notebook window titled "jupyter PDSA Week 1 Lecture 1" with the following code and output:

```
Compute primes from 1 to 100

In [1]: def factors(n):
        factorlist = []
        for i in range(1, n+1):
            if n%i == 0:
                factorlist.append(i)
        return factorlist

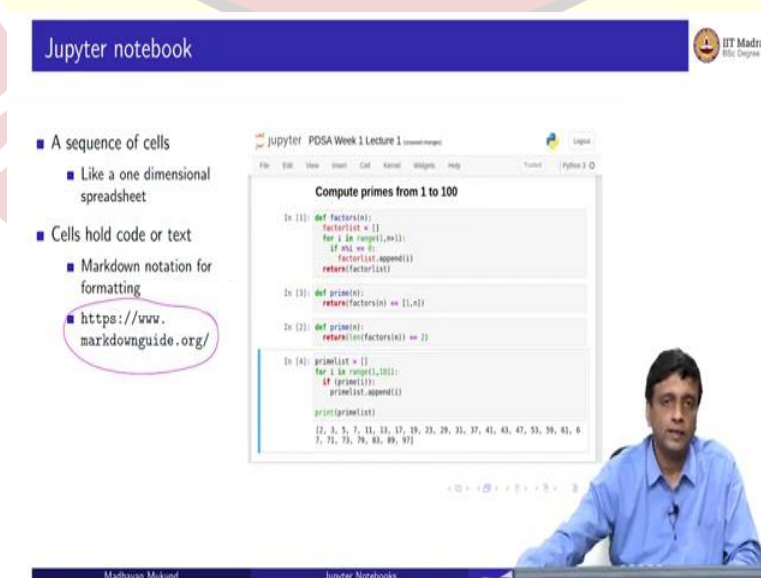
In [3]: def primes(n):
        return factors(n) == [1, n]

In [2]: def prime(n):
        return len(factors(n)) == 2

In [4]: primelist = []
        for i in range(1, 101):
            if prime(i):
                primelist.append(i)
        print(primelist)

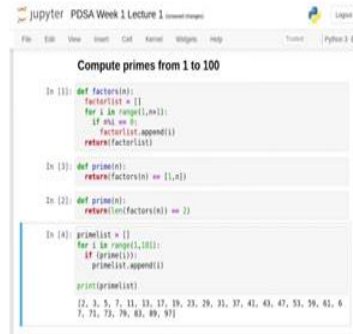
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]
```

The presenter, Madhavan Mukund, is visible in the bottom right corner.



The slide shows a Jupyter notebook titled "Jupyter notebook" with a "UT Madras" logo. It features a list of bullet points on the left: "A sequence of cells", "Like a one dimensional spreadsheet", "Cells hold code or text", "Markdown notation for formatting", and a link to "https://www.markdownguide.org/". The main area displays the same Jupyter notebook window as the previous slide, showing the code and output for computing primes. The presenter, Madhavan Mukund, is visible in the bottom right corner.

- A sequence of cells
 - Like a one dimensional spreadsheet
- Cells hold code or text
 - Markdown notation for formatting
 - <https://www.markdownguide.org/>
- Edit and re-run individual cells to update environment



```

In [1]: def factors(n):
        factorlist = []
        for i in range(1,n+1):
            if n%i == 0:
                factorlist.append(i)
        return factorlist

In [2]: def prime(n):
        return factors(n) == [1,n]

In [3]: def primes():
        primelist = []
        for i in range(1,101):
            if prime(i):
                primelist.append(i)
        return primelist

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

```



So, the concrete interface that we are going to look at is something called a Jupyter notebook. So, you may be familiar with a spreadsheet. So, what is a spreadsheet? A spreadsheet is basically a large, in some sense an indefinitely large square matrix of cells and in each of these cells you can put a value, you can put some text, you can put a formula.

So, it is unstructured in the sense that it does not tell you specifically what each cell should contain, but it is structured in the sense that there is a position, each cell has a kind of location you have a column number and a row number and then you can write formula saying this cell should be the sum of two other cells or it should be the sum of a column of cells or a row of cells and so on.

So, Jupyter notebook is like a spreadsheet except you have only this one column, so you do not have multiple columns but you can see that you have one cell, then another cell, then another cell, then another cell, so you have a sequence of cells from top to bottom, so that is what a spreadsheet looks like.

Now, in a spreadsheet you can do whatever you want with a cell. You can put many things, you can put text, you can put numbers, you could put formulas, you can put even charts, you can put diagrams and so on, so in a Jupyter notebook each cell holds either a piece of code or it holds some text which you want to insert to explain something about the code to somebody else. So, it is either code or text.

And as I said before the text is not just a comment as you would put in a Python program, it is something that has formatting. So, how do you specify the formatting? Well, the Jupyter

notebook supports a format called Markdown. So, this is a very simple type of format, which is like the type of formatting you would do if you were just composing some text in a normal text mode without a word processor.

So, you would basically have a hyphen to indicate a bulleted item and all that and the thing with Markdown is that it will convert it into nice formatted output. So, there are various resources on the internet, so here is one, which will tell you how to actually use Markdown to generate formatted output, so I will leave you to look that up, it is not very important for this particular course, except that it can be done.

So, the main thing about a Jupyter notebook is that like in a spreadsheet, we can dynamically change the spreadsheets contents by either updating a cell with a new value or when we update some values we can rerun some formulas either explicitly or implicitly to recompute some other values. So, in the same way in a Jupyter notebook you can add, code or update code and then rerun that code. So, you can change definitions of functions, you can rerun an output and see how it changes with each edit.

(Refer Slide Time: 9:30)

Jupyter notebook ...

- Supports different kernels
 - Julia, Python, R
 - We will use it only for Python
- Widely used to document and disseminate ML projects
 - Solutions to problems posed on platforms like Kaggle <https://www.kaggle.org>
- ACM Software Systems Award 2017

```
In [1]: import matplotlib.pyplot as plt
import numpy as np

# Data for plotting
t = np.arange(0, 2.0, 0.01)
s = 1 + np.sin(2 * np.pi * t)

fig, ax = plt.subplots()
ax.plot(t, s)

plt.show()
```

Madhavan Mukund Jupyter Notebooks

So, you might wonder about the name Jupyter, so the notebook as such, this is a spreadsheet like format, this notebook as such is not specific to Python, so it is a generic interface to any programming environment, which allows you to kind of interpret your code where you can write code and then immediately get it to be recognized and run.

So, that is what happens in Python, we change the code and then automatically the python interpreter is able to read the updated code and run it. So, Jupyter was designed to work with three different languages, Julia, Python and R. So, R you may have come across in some statistical context, so Julia is also a kind of scripting language for these kinds of calculations.

So, if you take the first few letters of Julia, Python and R you get Jupyter and that is the origin of the name. We of course, will not be using it for anything other than Python, but it is important to note that Jupyter as a framework is not limited to only Python, you can use different what they call kernels, you can have different programming languages supported behind the notebook as to the code that you run.

So, one interesting, one reason we are focusing on the Jupyter notebook format is not that it is necessary in some sense to understand data structures and algorithms in this course, but it is an extremely popular format in machine learning, which is the broader scope of this whole program. So, if you look at code which is available for many machine learning projects, it is typically saved and disseminated using the Jupyter notebook format.

In particular if about the site called Kaggle, so Kaggle is a competition for ML – Machine Learning, where they post problems, so some of these are synthetic problems, some of them are real problems, some of them even have prize money because somebody wants a problem solved and they are looking, it is like crowd sourcing, they want a number of people to attempt it.

And after the competition closes many people will post their solutions and these solutions are typically in this notebook format, so you can then access the solution which will have both the code and the documentation and then you can examine it, rerun it, tinker with it and so on. So, in particular this means that you get the opportunity to take somebody else's code for a given problem and see how it works by modifying it and trying to enhance it.

So, the Jupyter notebook has become so successful, especially with the growth of interest in Machine Learning that it has won a number of awards, so the Jupyter project which is behind the Jupyter notebook, one in particular this ACM Software Systems award in 2017, which is given for Significant Software Systems, which are used by the community. So, it is a fairly powerful tool.

(Refer Slide Time: 12:18)

So, of course you can run Jupyter notebook on an individual system, whatever system you have, whether it is Windows or Mac or Linux by installing it and like any other Python system you also have to import and install the relevant libraries that you need. We on the other hand will be using a publicly available form of the Jupyter notebook which is put up by Google in what is called Co-lab, which is short for the Co laboratory.

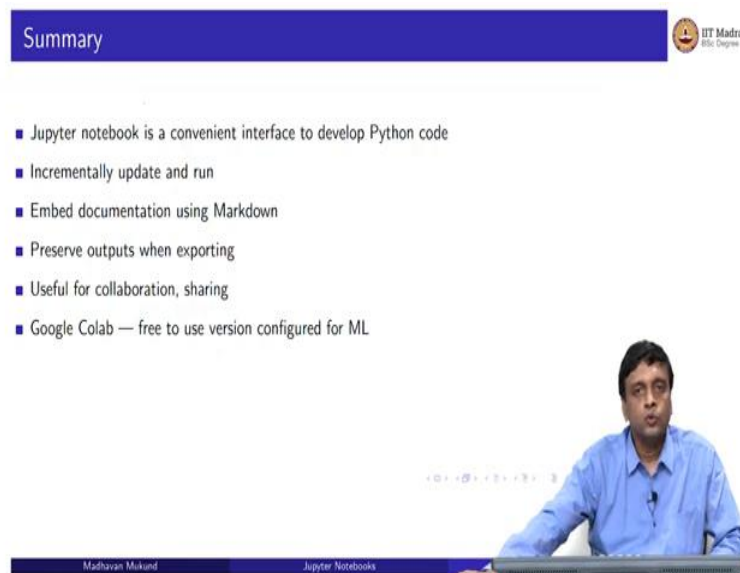
So, colab.research.google.com leads you to an interface where you can create these notebooks and save them and it is most importantly free to use. So, this has a slightly different look and feel from the Jupyter notebook that you would install on your own system, but essentially it is the same broad structure, which has been customized by Google for their internal use and then released for public use.

So, it is a customized Jupyter notebook. And one of the things which will be useful for you, not necessarily in this course but outside this course and the other courses that you are doing is that Colab has all the standard packages for machine learning pre-loaded. So, in particular there is a very popular pack library called scikit-learn, which has a lot of the standard machine learning models already implemented in it which you can call and use.

And there is Google's own library called TensorFlow which is used for deep learning or deep neural networks, plus because it is running on the cloud you also have access to hardware which is beyond the limitations of your personal computer. In particular Google makes it possible for you to run some of this machine learning code on what is called a GPU.

A GPU is a Graphical Processing Unit and it is very useful to run the kind of large scale matrix calculations which run behind the scenes in machine learning. So, many calculations which would take an enormous amount of time for you to run on the laptop will run in a much more reasonable amount of time on Colab using a GPU.

(Refer Slide Time: 14:19)



So, to summarize we will use Jupyter notebooks because it is a convenient interface to develop python code. In particular this ability to edit save and share the code is useful for me as an instructor also to be able to distribute the code that we discuss in this class, to you after the class. So, you can incrementally update and run your code. You can write documentation in between your code using this Markdown syntax.

And most importantly as I said you can preserve the state of the notebook, in terms of what you have run, what outputs you have generated and export it. And this is extremely useful for collaboration and for sharing, whether you are sharing it with a colleague or you are sharing it for teaching purposes like we are doing here. And the particular version of the Jupyter notebook that we are going to be using is Google's Colab which is free to use and it is configured for Machine Learning.