The African School of Fundamental Physics and Applications

Integrating Scientific Computing into Math and Science Classes

Dave Biersach
Brookhaven National Laboratory
dbiersach@bnl.gov

Session 01

Algebra, Statistics, and Trigonometry

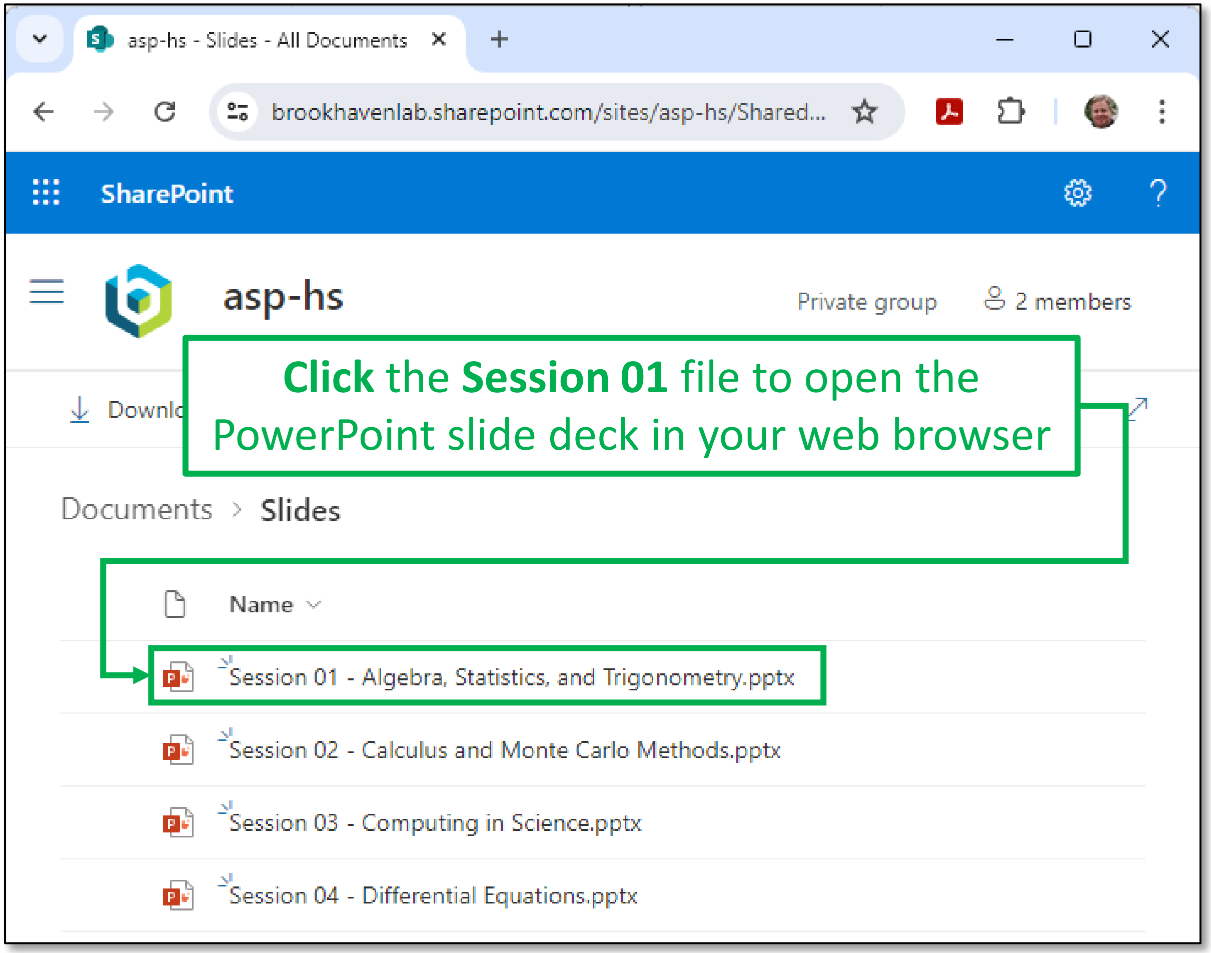
Brookhaven National Laboratory

Welcome!

- My name is **Dave Biersach**
- I am a Senior Technology Architect at BNL
- I am a 1989 graduate of the United States Military Academy, and I served in the 1991 Persian Gulf War as a Combat Engineer Officer
- I received a Ph.D. in Computational Physics at the Naval Postgraduate School in California
- I have worked for decades at **Microsoft** & **Pfizer**
- I have been married for 33 years, have three adult children, and have taught teachers and professors for the past 15 years

Get the Slides

Click on this link: ASP-HS Slides



asp-hs - Slides - All Documents
brookhavenlab.sharepoint.com/sites/asp-hs/Shared...
SharePoint
asp-hs
Private group
2 members
Downlo

Click the **Session 01** file to open the PowerPoint slide deck in your web browser

Documents > Slides

Name

Session 01 - Algebra, Statistics, and Trigonometry.pptx

Session 02 - Calculus and Monte Carlo Methods.pptx

Session 03 - Computing in Science.pptx

Session 04 - Differential Equations.pptx

The US Department of Energy

National Nuclear Safety Administration

Office of Science Laboratories

1 **Ames Laboratory**
Ames, Iowa

2 **Argonne National Laboratory**
Argonne, Illinois

3 **Brookhaven National Laboratory**
Upton, New York

4 **Fermi National Accelerator Laboratory**
Batavia, Illinois

5 **Lawrence Berkeley National Laboratory**
Berkeley, California

6 **Oak Ridge National Laboratory**
Oak Ridge, Tennessee

7 **Pacific Northwest National Laboratory**
Richland, Washington

8 **Princeton Plasma Physics Laboratory**
Princeton, New Jersey

9 **SLAC National Accelerator Laboratory**
Menlo Park, California

10 **Thomas Jefferson National Accelerator Facility**
Newport News, Virginia

Other DOE Laboratories

1 **Idaho National Laboratory**
Idaho Falls, Idaho

2 **National Energy Technology Laboratory**
Morgantown, West Virginia
Pittsburgh, Pennsylvania
Albany, Oregon

3 **National Renewable Energy Laboratory**
Golden, Colorado

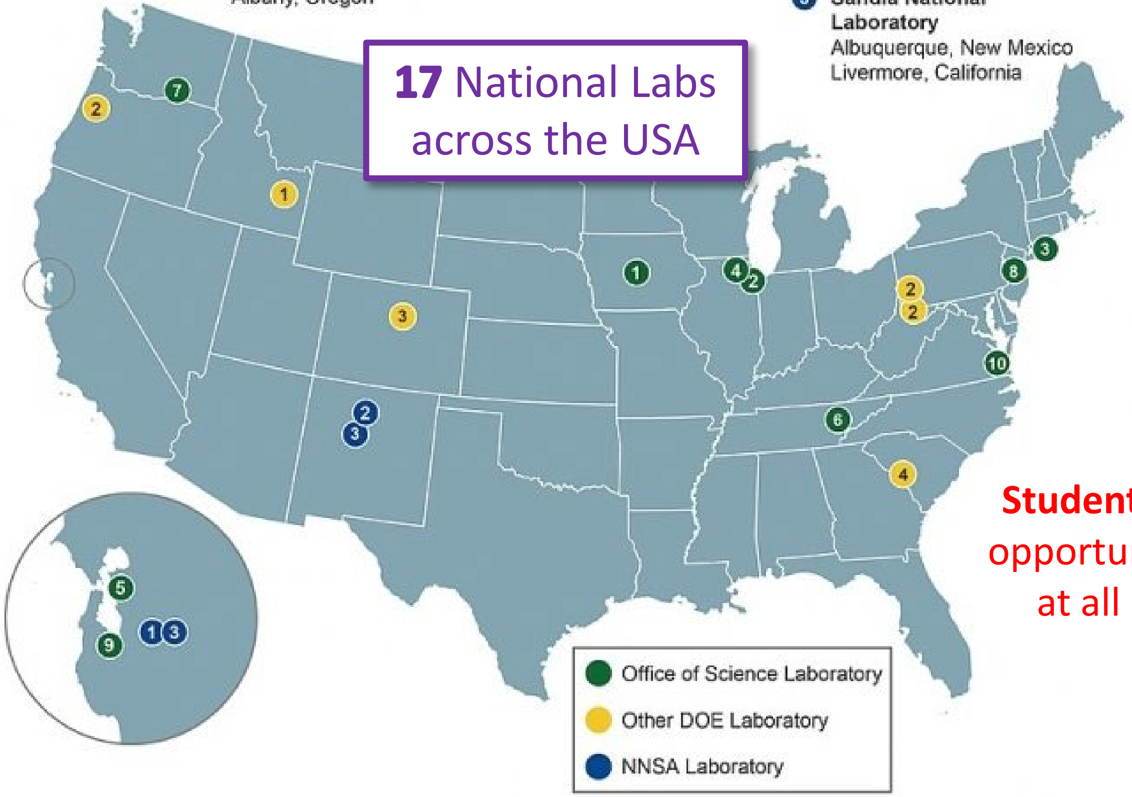
4 **Savannah River National Laboratory**
Aiken, South Carolina

NNSA Laboratories

1 **Lawrence Livermore National Laboratory**
Livermore, California

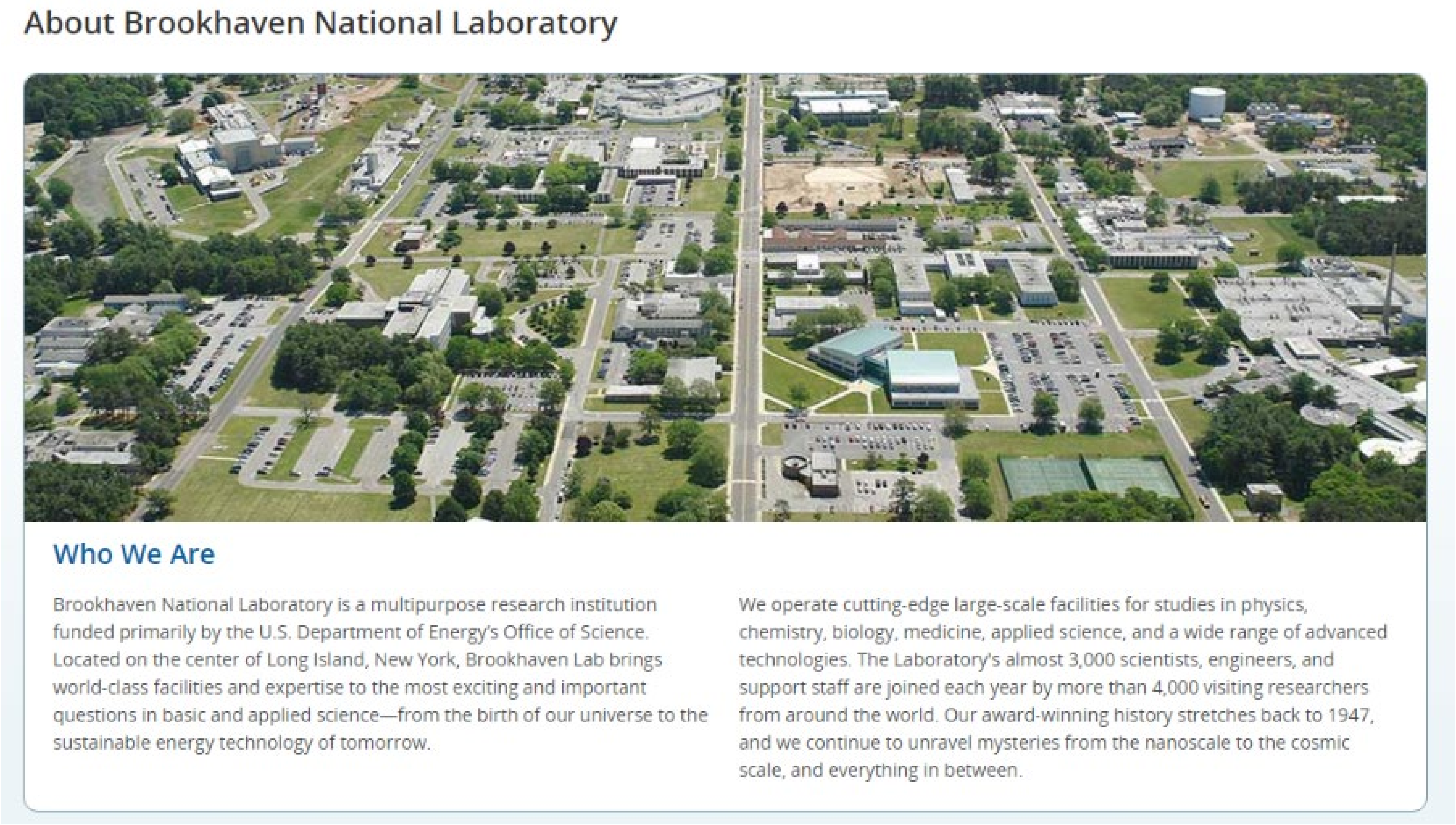
2 **Los Alamos National Laboratory**
Los Alamos, New Mexico

3 **Sandia National Laboratory**
Albuquerque, New Mexico
Livermore, California



17 National Labs across the USA

Student research opportunities exist at all US Labs



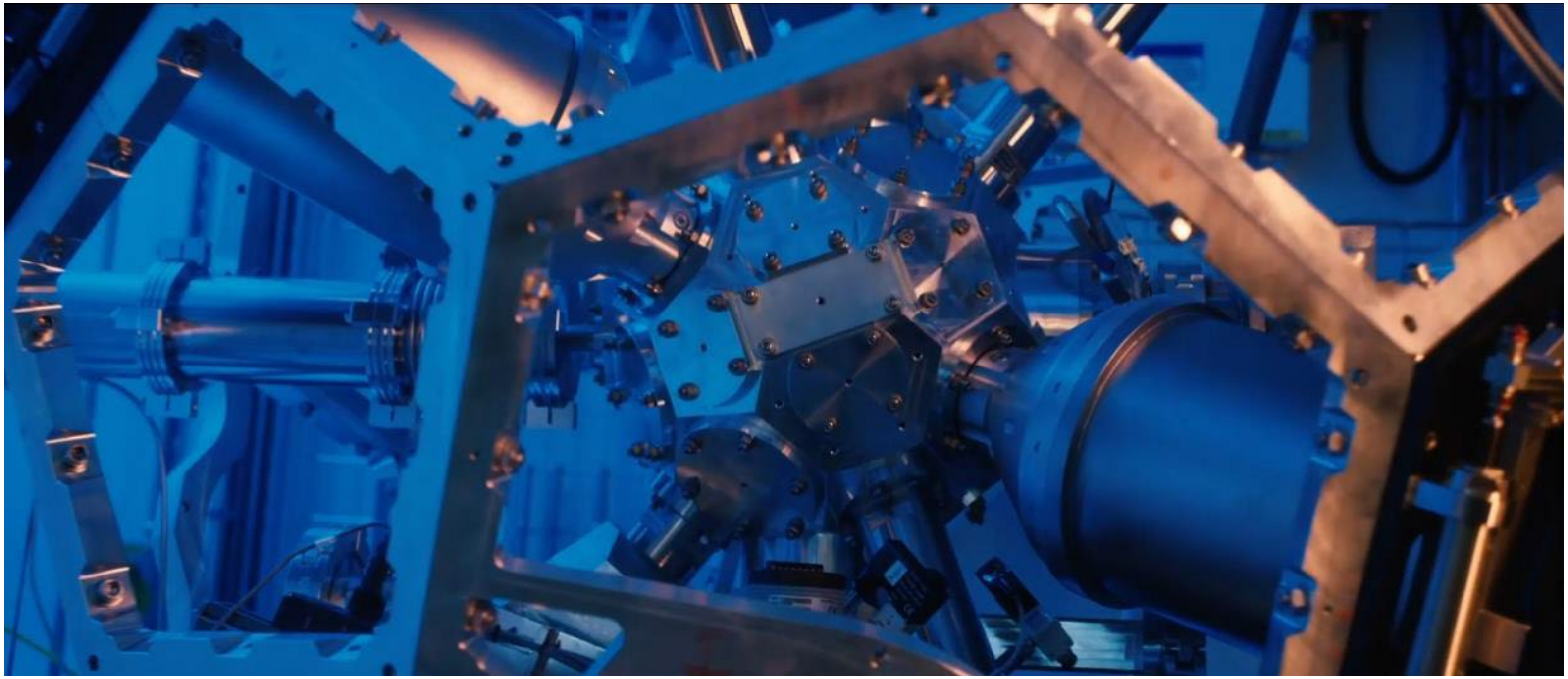
About Brookhaven National Laboratory

Who We Are

Brookhaven National Laboratory is a multipurpose research institution funded primarily by the U.S. Department of Energy's Office of Science. Located on the center of Long Island, New York, Brookhaven Lab brings world-class facilities and expertise to the most exciting and important questions in basic and applied science—from the birth of our universe to the sustainable energy technology of tomorrow.

We operate cutting-edge large-scale facilities for studies in physics, chemistry, biology, medicine, applied science, and a wide range of advanced technologies. The Laboratory's almost 3,000 scientists, engineers, and support staff are joined each year by more than 4,000 visiting researchers from around the world. Our award-winning history stretches back to 1947, and we continue to unravel mysteries from the nanoscale to the cosmic scale, and everything in between.

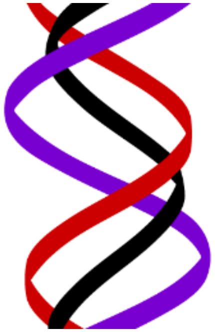
About Brookhaven National Laboratory



YouTube - This is Brookhaven Lab

What is Scientific Computing?

- Scientific computing problems **cannot be solved** using just a graphing calculator or a spreadsheet program
    - A computer should not be viewed as just another closed-form benchtop instrument with fixed functionality
    - SciComp does not require writing thousands of lines of code to answer problems – complete code usually fits on **one** slide!
- SciComp is **applied** computer science
    - The first name of CompSci is *computer*
    - The first name of SciComp is **science**
    - A **triple helix** of math, science, and computing



SciComp vs CompSci

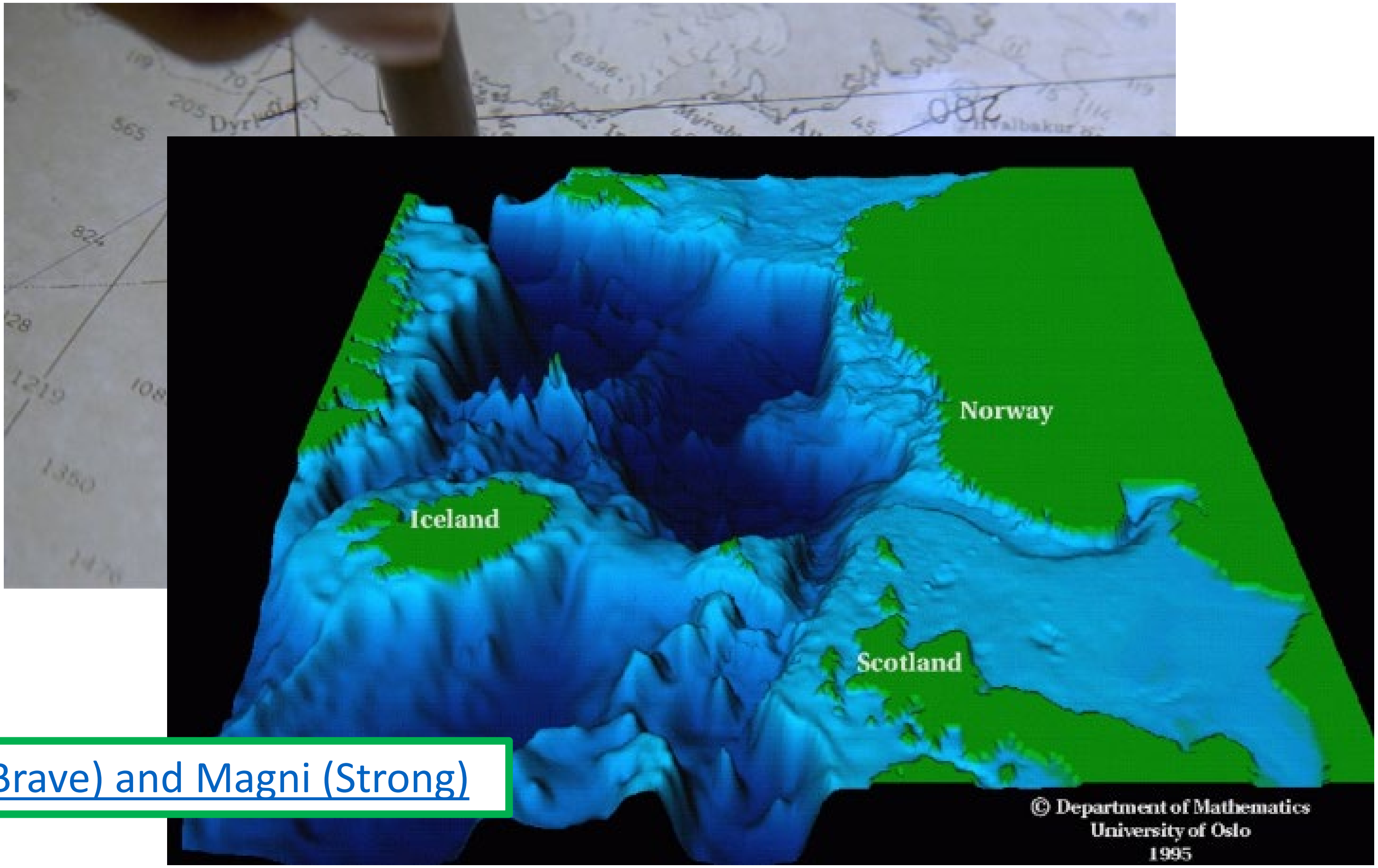
Scientific Computing

- Probability and Statistics
- Simulation and Modelling
- Data Visualization
- Storing and Analyzing Very Large Datasets
- Parallel & Distributed Algorithms
- Speed and Accuracy Paramount
- Functional Languages
- Open-Ended Problems with Unknown Solutions

Computer Science

- General Data Structures
- Design Methodologies
- Procedural Languages
- Stand-Alone Programs
- Emphasis on Object-Orientation
- Simple Data Models
- Sequential Algorithms
- Less Graphics Intensive
- Directed Closed-Form Problems with Known Solutions

Example SciComp Topic
Multidimensional Interpolation

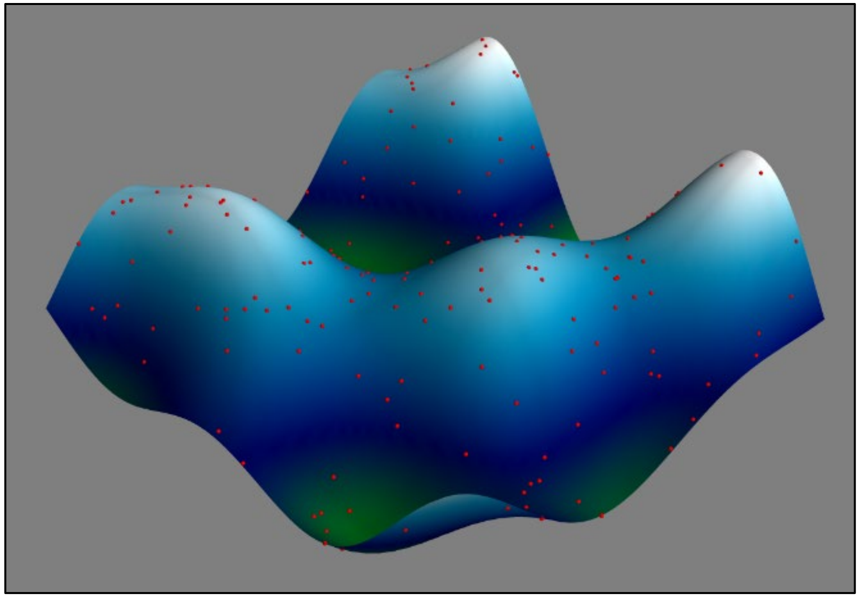


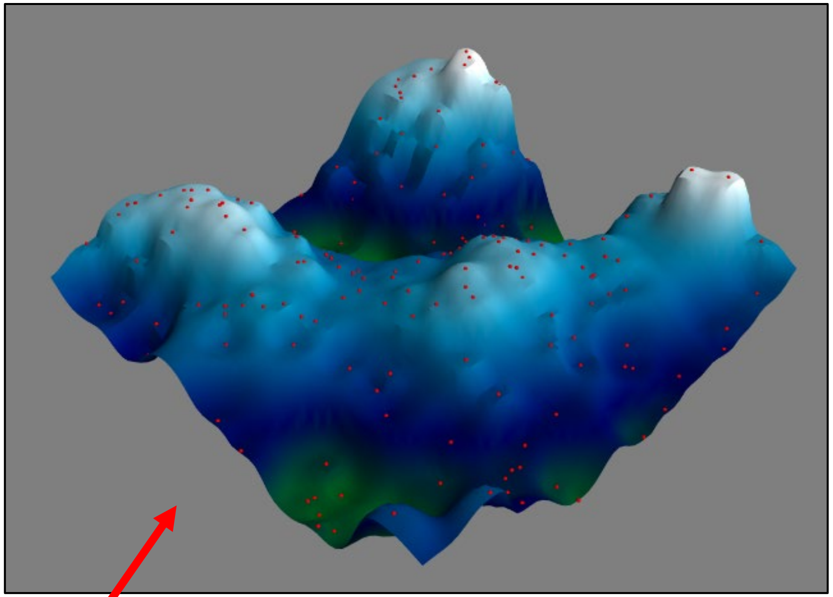
© Department of Mathematics
University of Oslo
1995

Modi (Brave) and Magni (Strong)

Example SciComp Topic

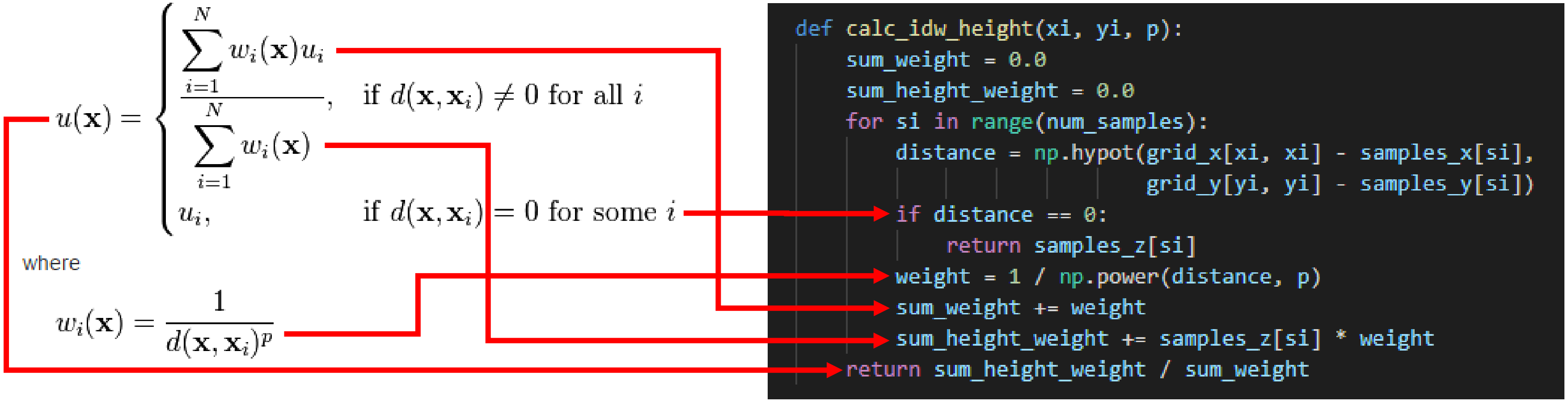
Multidimensional Interpolation





A first order 3-D approximation of the ocean floor based upon only 220 sample (red) points (sonar timings)

SciComp As Translational Science



$$u(\mathbf{x}) = \begin{cases} \dfrac{\sum_{i=1}^{N} w_i(\mathbf{x}) u_i}{\sum_{i=1}^{N} w_i(\mathbf{x})}, & \text{if } d(\mathbf{x},\mathbf{x}_i) \neq 0 \text{ for all } i \\ u_i, & \text{if } d(\mathbf{x},\mathbf{x}_i) = 0 \text{ for some } i \end{cases}$$

where

$$w_i(\mathbf{x}) = \frac{1}{d(\mathbf{x},\mathbf{x}_i)^p}$$

```python
def calc_idw_height(xi, yi, p):
    sum_weight = 0.0
    sum_height_weight = 0.0
    for si in range(num_samples):
        distance = np.hypot(grid_x[xi, xi] - samples_x[si],
                            grid_y[yi, yi] - samples_y[si])
        if distance == 0:
            return samples_z[si]
        weight = 1 / np.power(distance, p)
        sum_weight += weight
        sum_height_weight += samples_z[si] * weight
    return sum_height_weight / sum_weight
```

11 lines of code can change the world!

SciComp is the ability to translate mathematical expressions of scientific concepts into correct and efficient software code

SciComp 101

Foundations of Scientific Computing

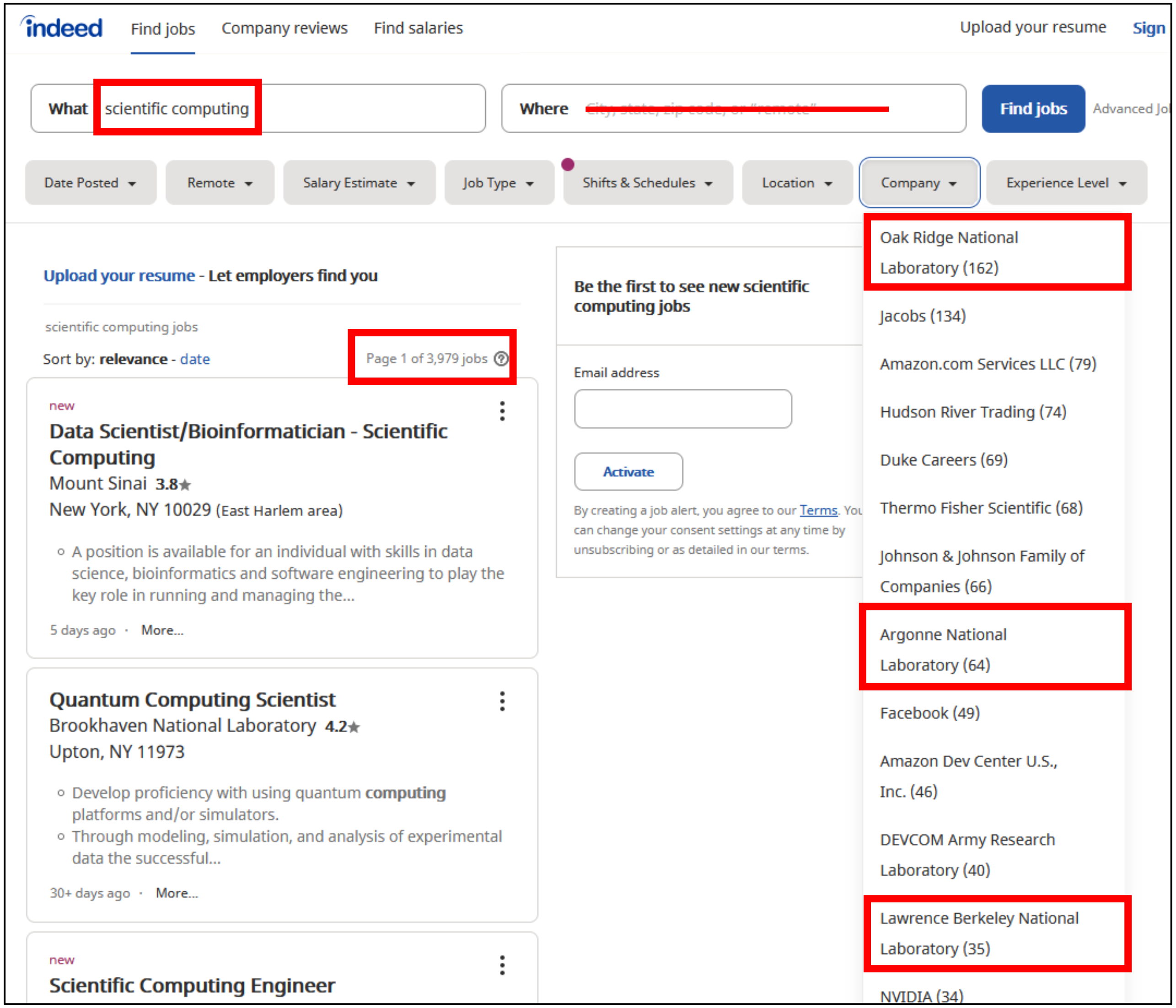
- Packaged as **20** high school lessons with hands-on student programming labs using the **free Google Colab service**
  - BNL provides all required presentations, sample code, lab exercises, and teacher guide
  - The software tools are 100% open-source and free of charge
  - The students can use Windows, Apple Mac, or **Chromebooks**
- The lessons are split into **three 20-minute sections**
  - The last 20-minute section in each session is optional & not required for pedagogical continuity
  - This structure enables sessions to be delivered within a high school science or math course **if limited to a 40-minute class period**

SciComp 101

Foundations of Scientific Computing

- Objectives
  - Provide patterns for solving real-world **science problems** by writing custom software
  - Demonstrate how **scientific computing impacts all science disciplines**
  - Enable students to **translate scientific formulas into correct and efficient code**
  - Review techniques for the **effective visualization** of complex data
  - Show optimal methods to store and analyze very large data sets
  - **Prepare students to conduct interdisciplinary research at world-class institutions**

SciComp = The Pathway to Internships



indeed Find jobs Company reviews Find salaries Upload your resume Sign

What scientific computing

Where ~~City, state, zip code, or "remote"~~

Find jobs Advanced Jo

Date Posted ▾ Remote ▾ Salary Estimate ▾ Job Type ▾ Shifts & Schedules ▾ Location ▾ Company ▾ Experience Level ▾

Upload your resume - Let employers find you

scientific computing jobs

Sort by: **relevance** - date

Page 1 of 3,979 jobs

new

Data Scientist/Bioinformatician - Scientific Computing

Mount Sinai 3.8★

New York, NY 10029 (East Harlem area)

- A position is available for an individual with skills in data science, bioinformatics and software engineering to play the key role in running and managing the...

5 days ago · More...

Quantum Computing Scientist

Brookhaven National Laboratory 4.2★

Upton, NY 11973

- Develop proficiency with using quantum **computing** platforms and/or simulators.
- Through modeling, simulation, and analysis of experimental data the successful...

30+ days ago · More...

new

Scientific Computing Engineer

Be the first to see new scientific computing jobs

Email address

Activate

By creating a job alert, you agree to our Terms. You can change your consent settings at any time by unsubscribing or as detailed in our terms.

Oak Ridge National Laboratory (162)

Jacobs (134)

Amazon.com Services LLC (79)

Hudson River Trading (74)

Duke Careers (69)

Thermo Fisher Scientific (68)

Johnson & Johnson Family of Companies (66)

Argonne National Laboratory (64)

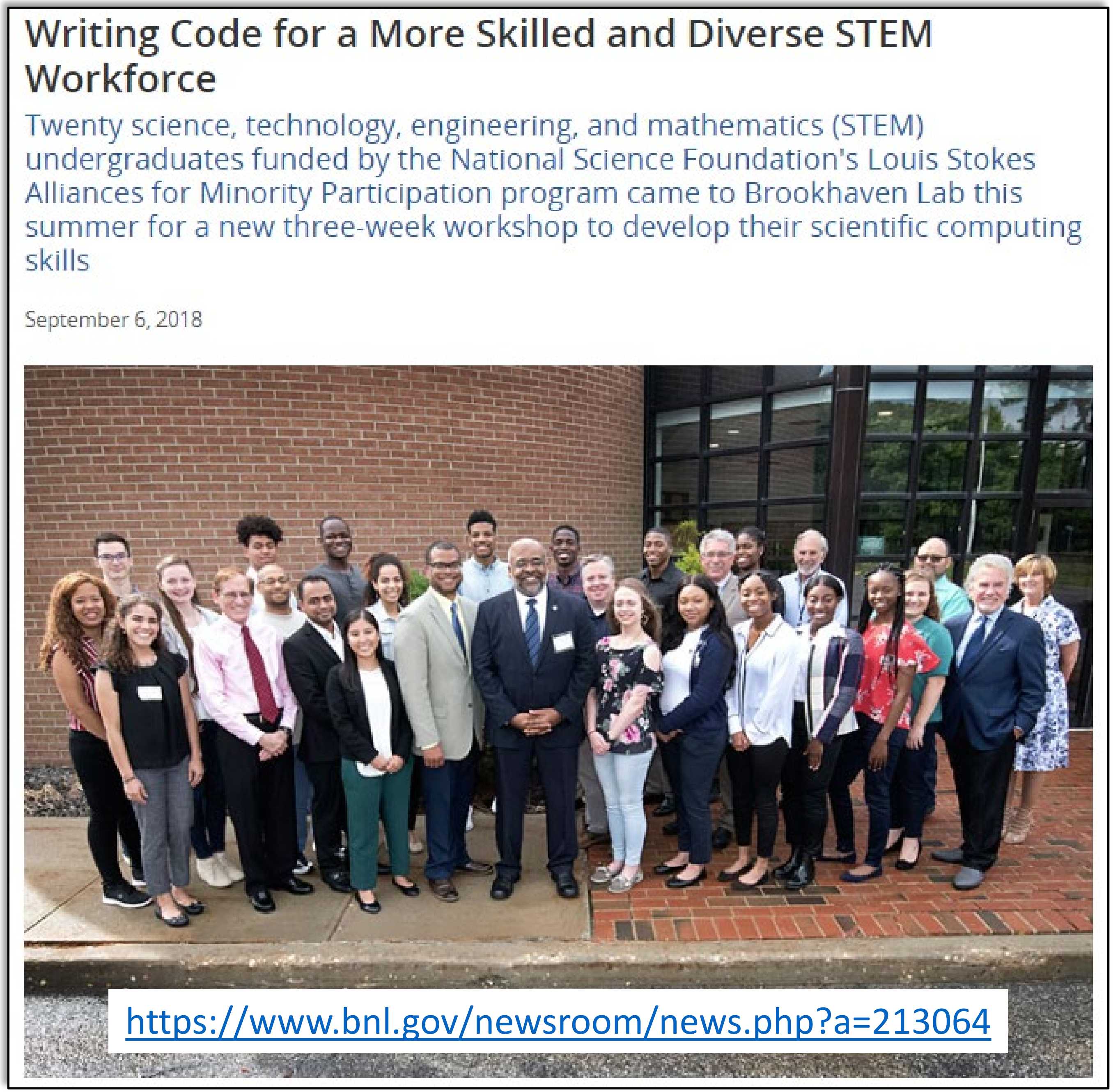
Facebook (49)

Amazon Dev Center U.S., Inc. (46)

DEVCOM Army Research Laboratory (40)

Lawrence Berkeley National Laboratory (35)

NVIDIA (34)



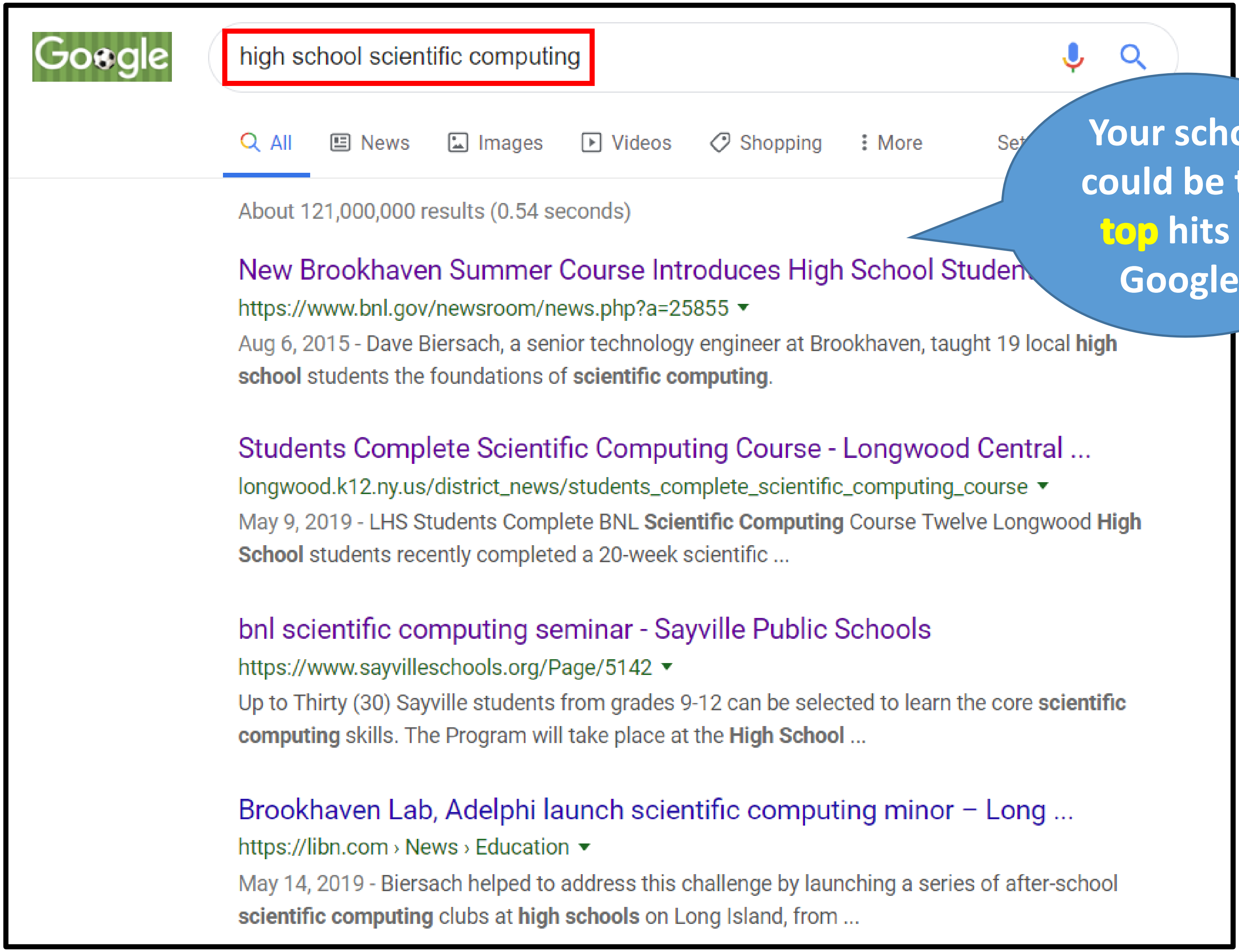
Writing Code for a More Skilled and Diverse STEM Workforce

Twenty science, technology, engineering, and mathematics (STEM) undergraduates funded by the National Science Foundation's Louis Stokes Alliances for Minority Participation program came to Brookhaven Lab this summer for a new three-week workshop to develop their scientific computing skills

September 6, 2018

https://www.bnl.gov/newsroom/news.php?a=213064

You can lead the world!



high school scientific computing

About 121,000,000 results (0.54 seconds)

New Brookhaven Summer Course Introduces High School Studen
https://www.bnl.gov/newsroom/news.php?a=25855
Aug 6, 2015 - Dave Biersach, a senior technology engineer at Brookhaven, taught 19 local **high school** students the foundations of **scientific computing**.

Students Complete Scientific Computing Course - Longwood Central ...
longwood.k12.ny.us/district_news/students_complete_scientific_computing_course
May 9, 2019 - LHS Students Complete BNL **Scientific Computing** Course Twelve Longwood **High School** students recently completed a 20-week scientific ...

bnl scientific computing seminar - Sayville Public Schools
https://www.sayvilleschools.org/Page/5142
Up to Thirty (30) Sayville students from grades 9-12 can be selected to learn the core **scientific computing** skills. The Program will take place at the **High School** ...

Brookhaven Lab, Adelphi launch scientific computing minor – Long ...
https://libn.com › News › Education
May 14, 2019 - Biersach helped to address this challenge by launching a series of after-school **scientific computing** clubs at **high schools** on Long Island, from ...

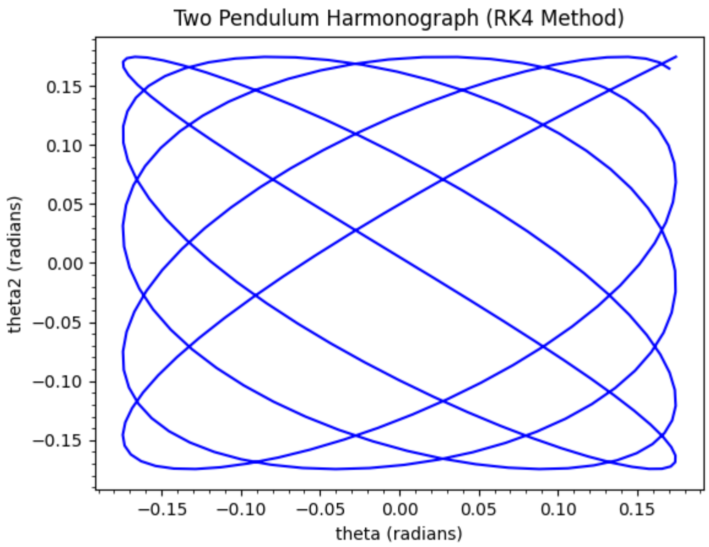
Your school could be the **top** hits in Google!

Mathematical Concepts

- Systems of Equations
- Probability Distributions
- Combinatorics
- Simulation & Modeling
- Monte Carlo Integration
- Polar & Spherical Coordinates
- Dynamical Systems
- Mesh Interpolation
- 2D Affine Transformations
- Vector & Complex Algebra
- Signals Analysis

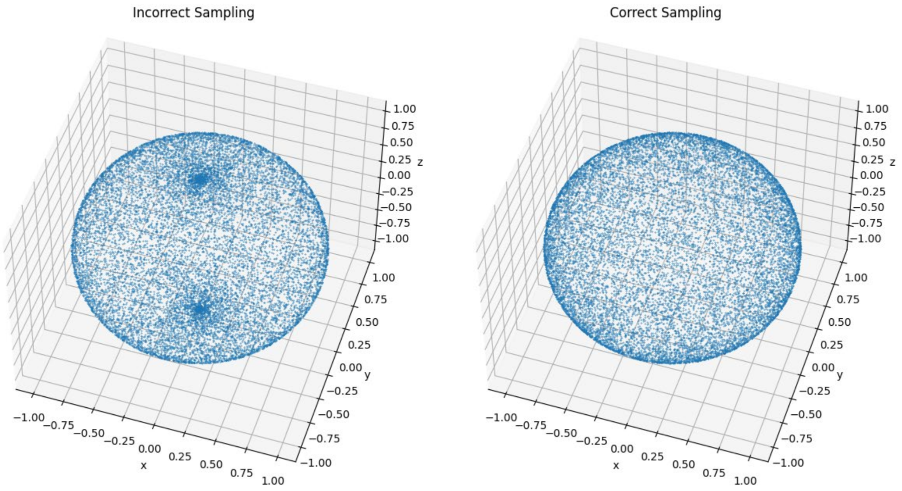
$$\boxed{\varphi = [1; \{1\}]}$$

$$\boxed{\sum_{n=1}^{\infty} \frac{1}{n^2} = 1 + \frac{1}{4} + \frac{1}{9} + \frac{1}{16} + \frac{1}{25} + \cdots = \frac{\pi^2}{6}}$$

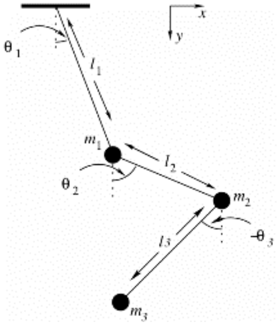


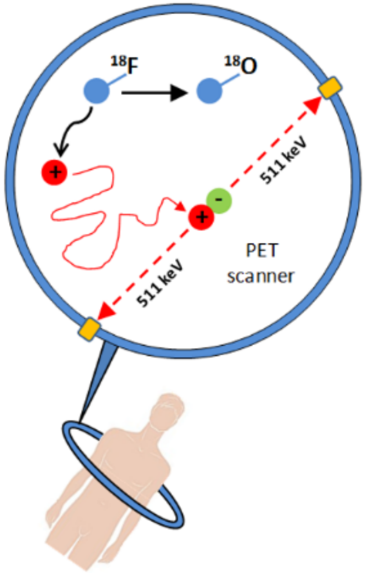
Computer Science Concepts

- Representations and Encodings
- Random Number Generation
- Strings, Arrays, Operators
- Loops, Functions, Recursion
- Searching & Sorting
- 2D and 3D Graphics
- Accuracy & Precision
- Runtime Complexity
- File I/O (CSV)

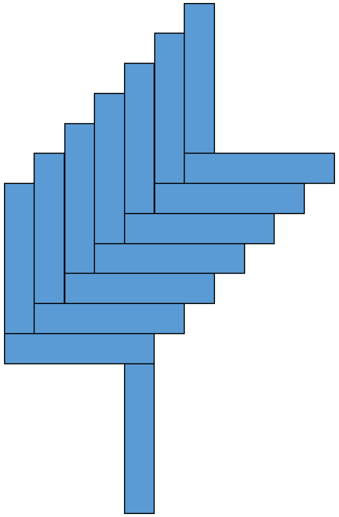


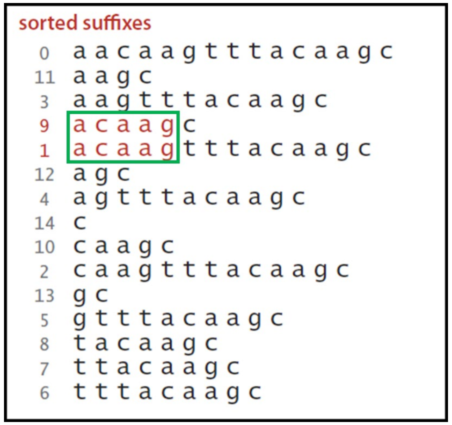
Science Concepts





- Mechanics and Kinematics
- Waves (Nyquist Sampling)
- Unit Conversion
- Genetic Sequence Analysis
- Balancing Ionic Equations
- Projectile Motion
- Equilibrium & Thermodynamics
- Radioactive Decay





sorted suffixes

```
 0  aacaagtttacaagc
11  aagc
 3  aagtttacaagc
 9  acaagc
 1  acaagtttacaagc
12  agc
 4  agtttacaagc
14  c
10  caagc
 2  caagtttacaagc
13  gc
 5  gtttacaagc
 8  tacaagc
 7  ttacaagc
 6  tttacaagc
```

$$\mathbf{10}P_2I_4 + \mathbf{13}P_4 + \mathbf{128}H_2O \rightarrow \mathbf{40}PH_4I + \mathbf{32}H_3PO_4$$

Phosphonium iodide

Scientific Computing with Python

- Python is quickly becoming one of the **most heavily used languages** in science projects
- Python runs on all major modern **operating systems** and is completely free and open-source (not vendor controlled)
- Python makes it easy for your code to directly integrate with a large spectrum of available 3rd party software
- Python code runs **consistently** on different platforms and scales well from small IoT devices to large server clusters
- Python benefits from a very active and growing user community that continues to enhance the language

Motivation

- Every **high school science research project** can benefit from even just a slight touch of **scientific computing**
  - Better **statistics** & data **visualization** on posters
  - Compelling analysis from modeling & simulation
  - Novel integration of computation is a big ***differentiator!***
- The lab exercises we have developed are taken directly from active research projects **at BNL**
  - We all learned how to ***read*** before we learned how to ***write*** - many junior BNL staff inherit existing code to fix or extend
  - **More than 80% of all summer research projects at BNL require high school interns to write code**

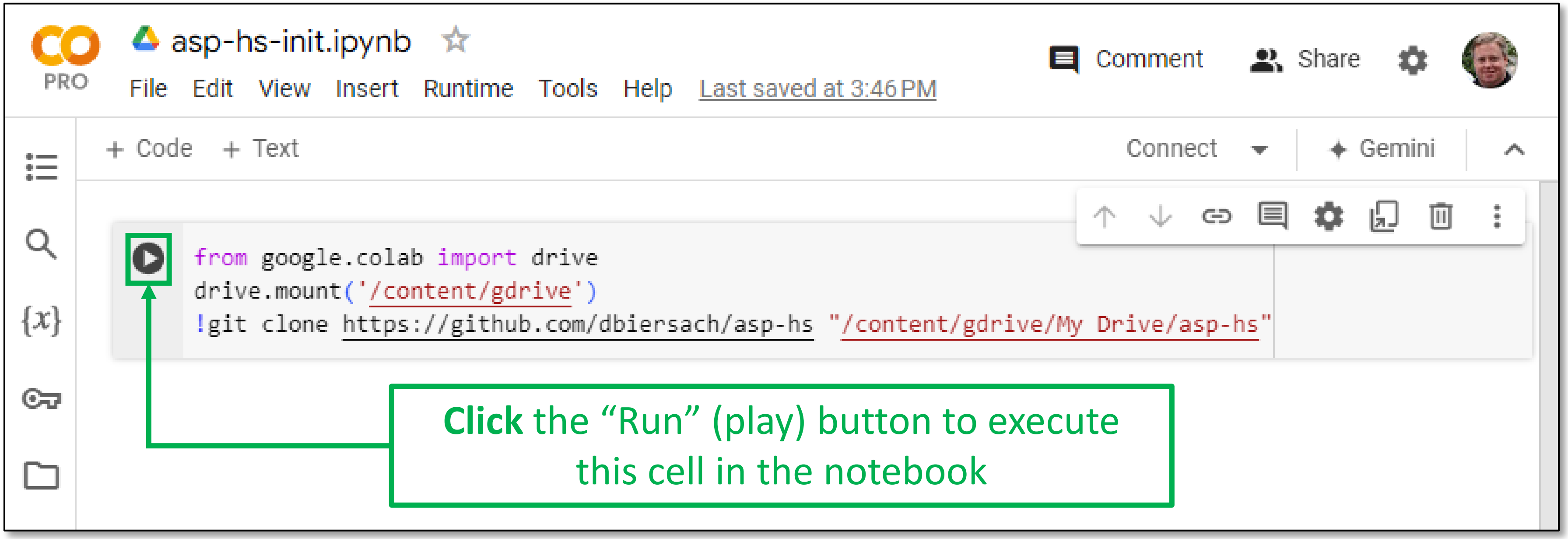
Motivation

- It does not take thousands of lines of code to keep importance science moving right along…
    - You don’t have to be a professional programmer or know all the arcane aspects of computer languages
    - The closer you get to **cutting edge science**, the less likely you’ll be able to just “download an app” to accomplish what you need
- If you don’t know how to code…
    - You will at some point start to ***subconsciously*** limit the types of analysis you can perform because you will remain at the mercy of the available software
    - Should software shape your science, or instead, will you shape software to **advance** your science?

Get the Courseware – **Step 1**

Click on this link: asp-hs-init.ipynb

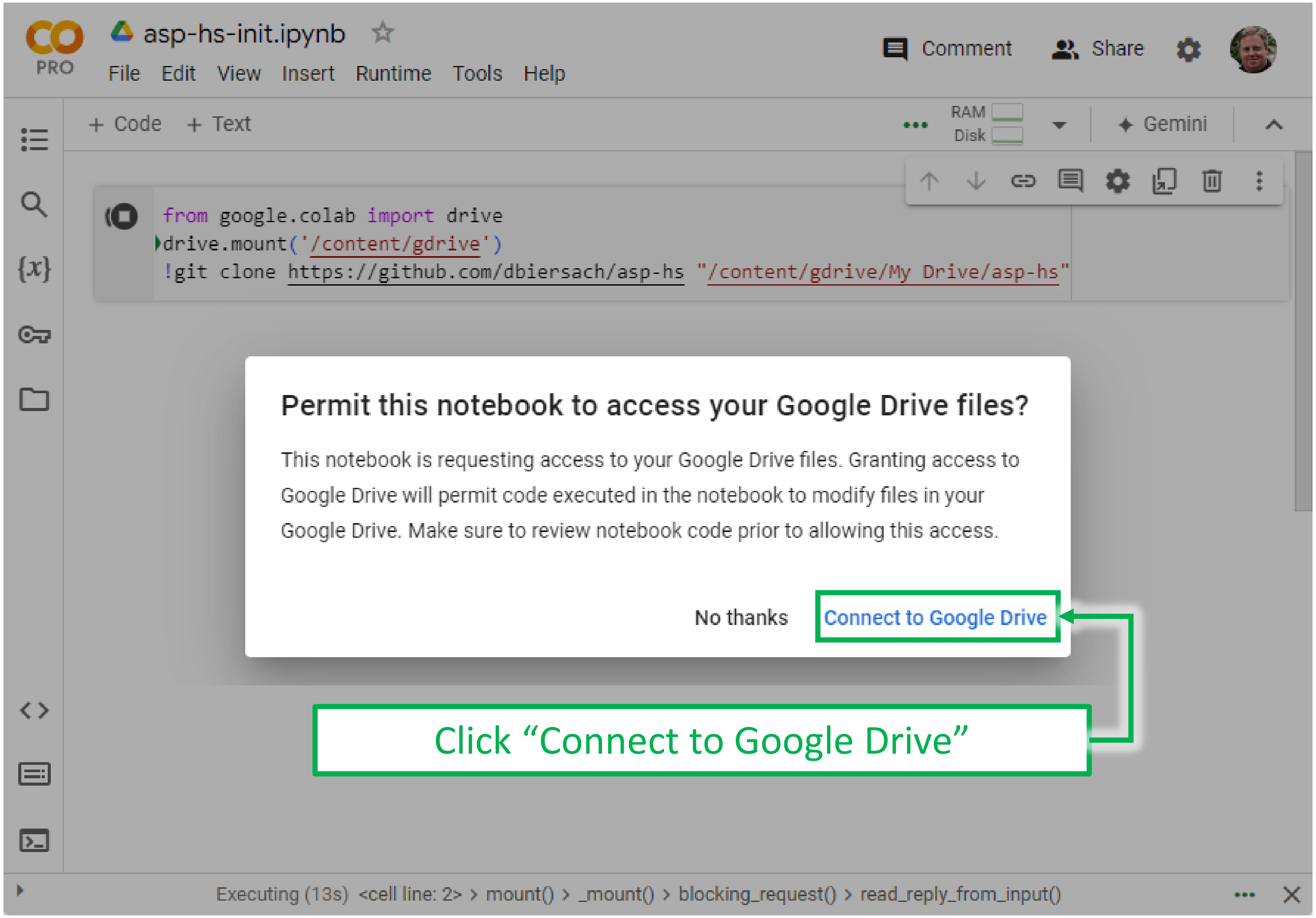
Your web browser should then display this page:



```
from google.colab import drive
drive.mount('/content/gdrive')
!git clone https://github.com/dbiersach/asp-hs "/content/gdrive/My Drive/asp-hs"
```

Click the "Run" (play) button to execute this cell in the notebook

Get the Courseware – **Step 2**



```
from google.colab import drive
drive.mount('/content/gdrive')
!git clone https://github.com/dbiersach/asp-hs "/content/gdrive/My Drive/asp-hs"
```

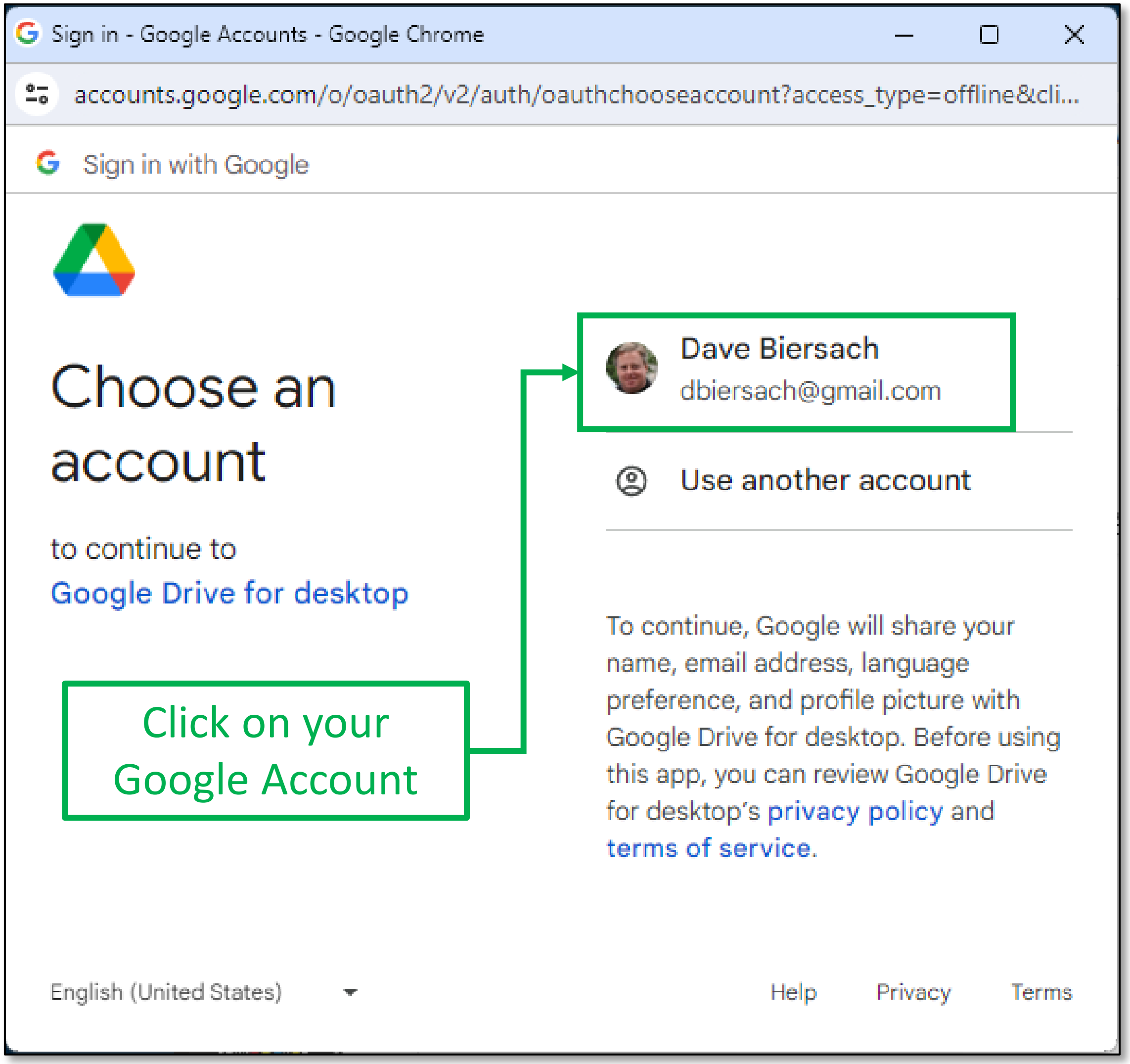
Permit this notebook to access your Google Drive files?

This notebook is requesting access to your Google Drive files. Granting access to Google Drive will permit code executed in the notebook to modify files in your Google Drive. Make sure to review notebook code prior to allowing this access.

Click “Connect to Google Drive”

Executing (13s) <cell line: 2> > mount() > _mount() > blocking_request() > read_reply_from_input()

Get the Courseware – **Step 3**



Sign in - Google Accounts - Google Chrome

accounts.google.com/o/oauth2/v2/auth/oauthchooseaccount?access_type=offline&cli...

Sign in with Google

Choose an account

to continue to

Google Drive for desktop

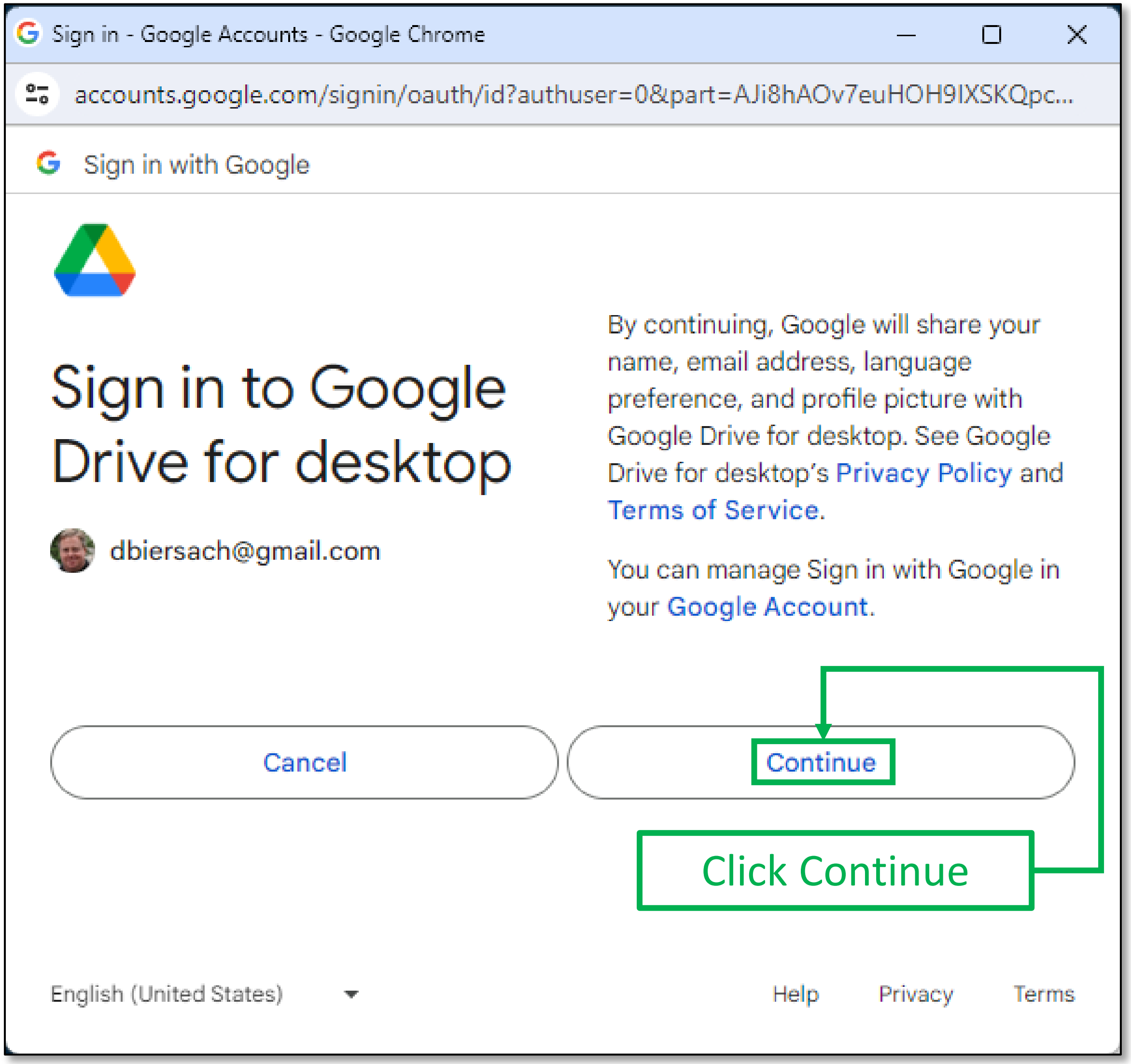
Dave Biersach
dbiersach@gmail.com

Use another account

To continue, Google will share your name, email address, language preference, and profile picture with Google Drive for desktop. Before using this app, you can review Google Drive for desktop's privacy policy and terms of service.

Click on your Google Account

Get the Courseware – **Step 4**



Sign in - Google Accounts - Google Chrome
accounts.google.com/signin/oauth/id?authuser=0&part=AJi8hAOv7euHOH9IXSKQpc...
Sign in with Google

Sign in to Google Drive for desktop

dbiersach@gmail.com

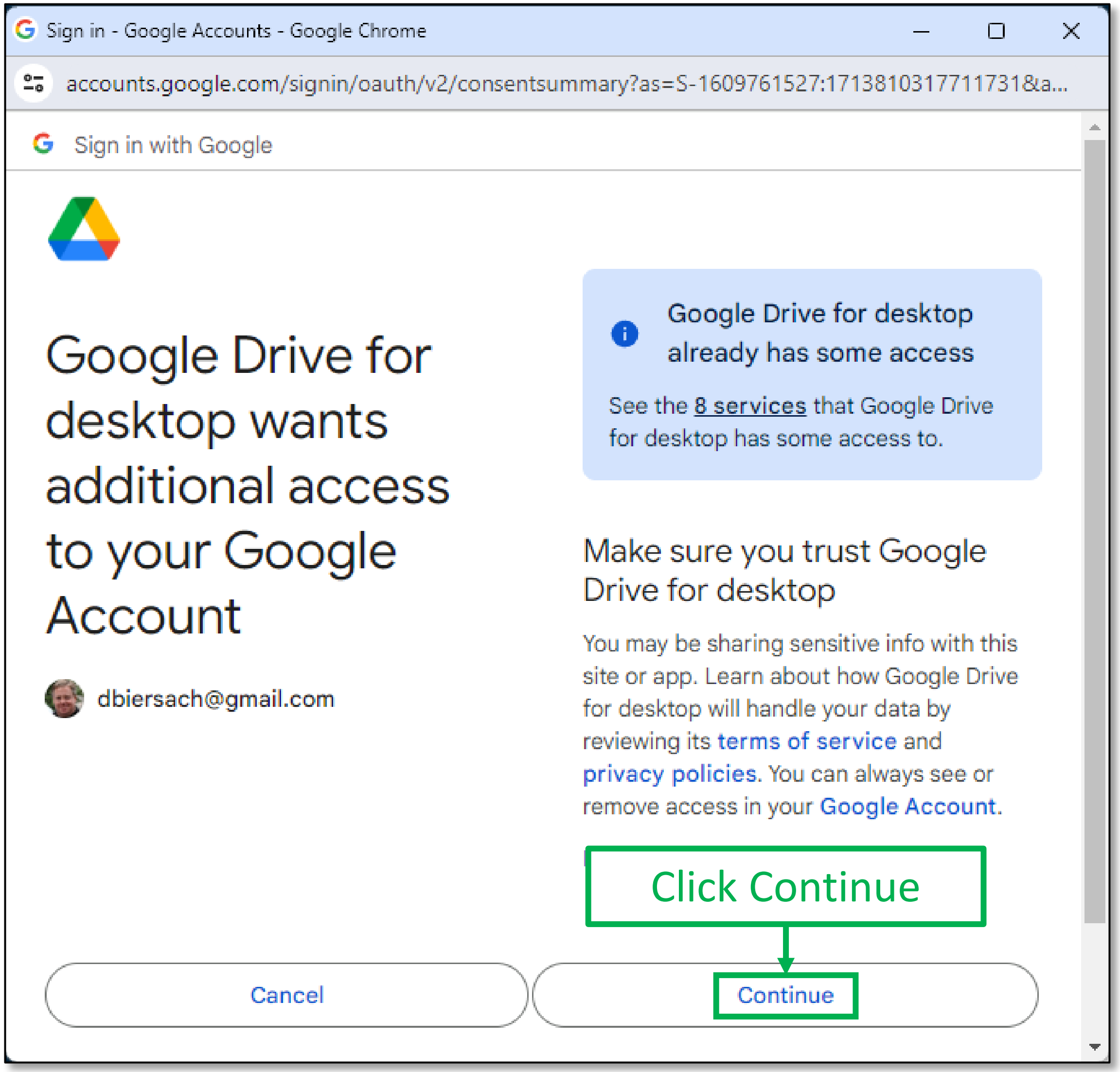
By continuing, Google will share your name, email address, language preference, and profile picture with Google Drive for desktop. See Google Drive for desktop's Privacy Policy and Terms of Service.

You can manage Sign in with Google in your Google Account.

Cancel
Continue
English (United States)
Help
Privacy
Terms

Click Continue

Get the Courseware – **Step 5**



Sign in - Google Accounts - Google Chrome
accounts.google.com/signin/oauth/v2/consentsummary?as=S-1609761527:1713810317711731&a...
Sign in with Google

Google Drive for desktop wants additional access to your Google Account

dbiersach@gmail.com

Google Drive for desktop already has some access

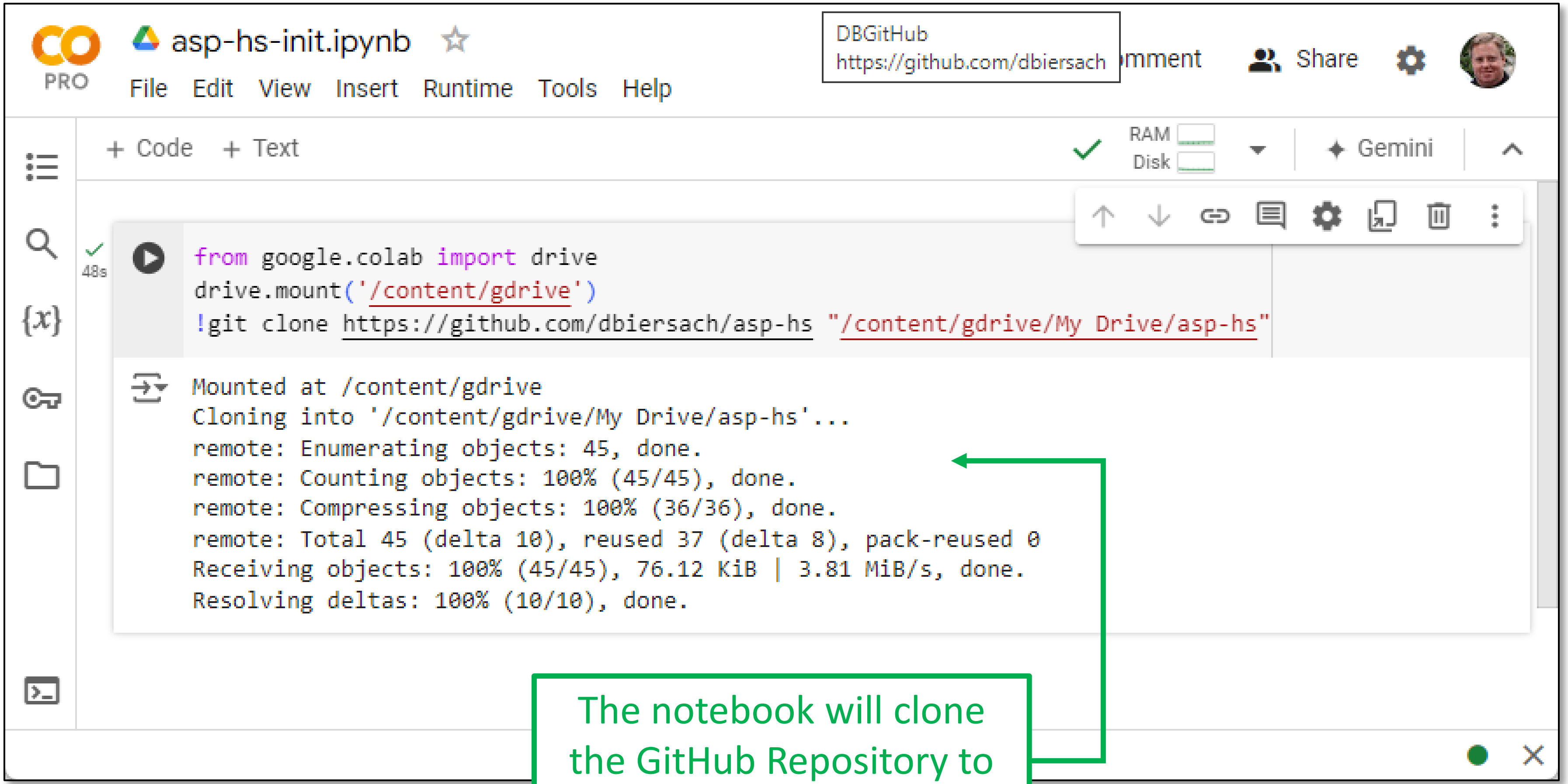
See the 8 services that Google Drive for desktop has some access to.

Make sure you trust Google Drive for desktop

You may be sharing sensitive info with this site or app. Learn about how Google Drive for desktop will handle your data by reviewing its terms of service and privacy policies. You can always see or remove access in your Google Account.

Click Continue

Get the Courseware – **Step 6**

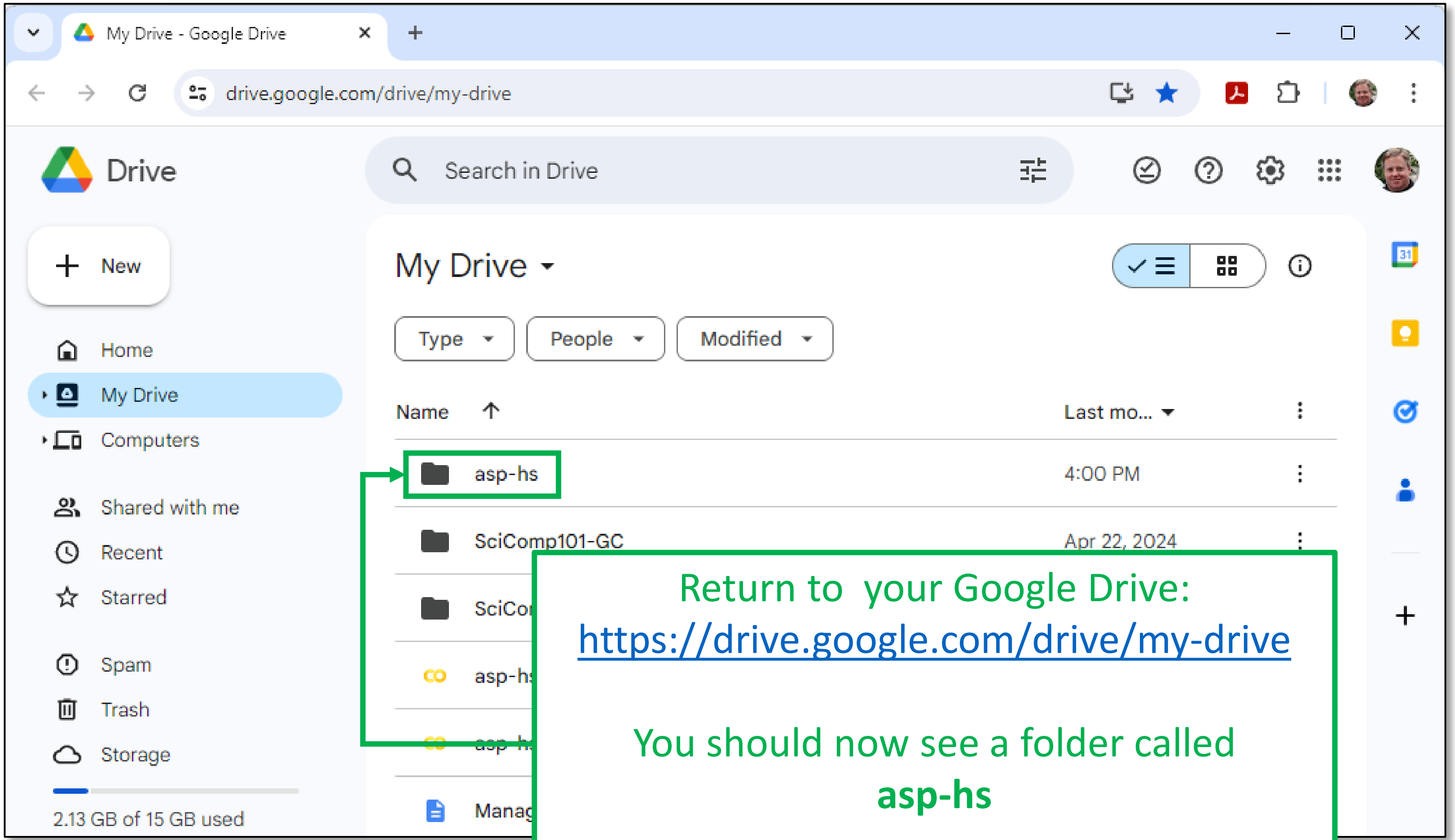


```
from google.colab import drive
drive.mount('/content/gdrive')
!git clone https://github.com/dbiersach/asp-hs "/content/gdrive/My Drive/asp-hs"
```

```
Mounted at /content/gdrive
Cloning into '/content/gdrive/My Drive/asp-hs'...
remote: Enumerating objects: 45, done.
remote: Counting objects: 100% (45/45), done.
remote: Compressing objects: 100% (36/36), done.
remote: Total 45 (delta 10), reused 37 (delta 8), pack-reused 0
Receiving objects: 100% (45/45), 76.12 KiB | 3.81 MiB/s, done.
Resolving deltas: 100% (10/10), done.
```

The notebook will clone the GitHub Repository to your Google Drive

Get the Courseware – **Step 7**

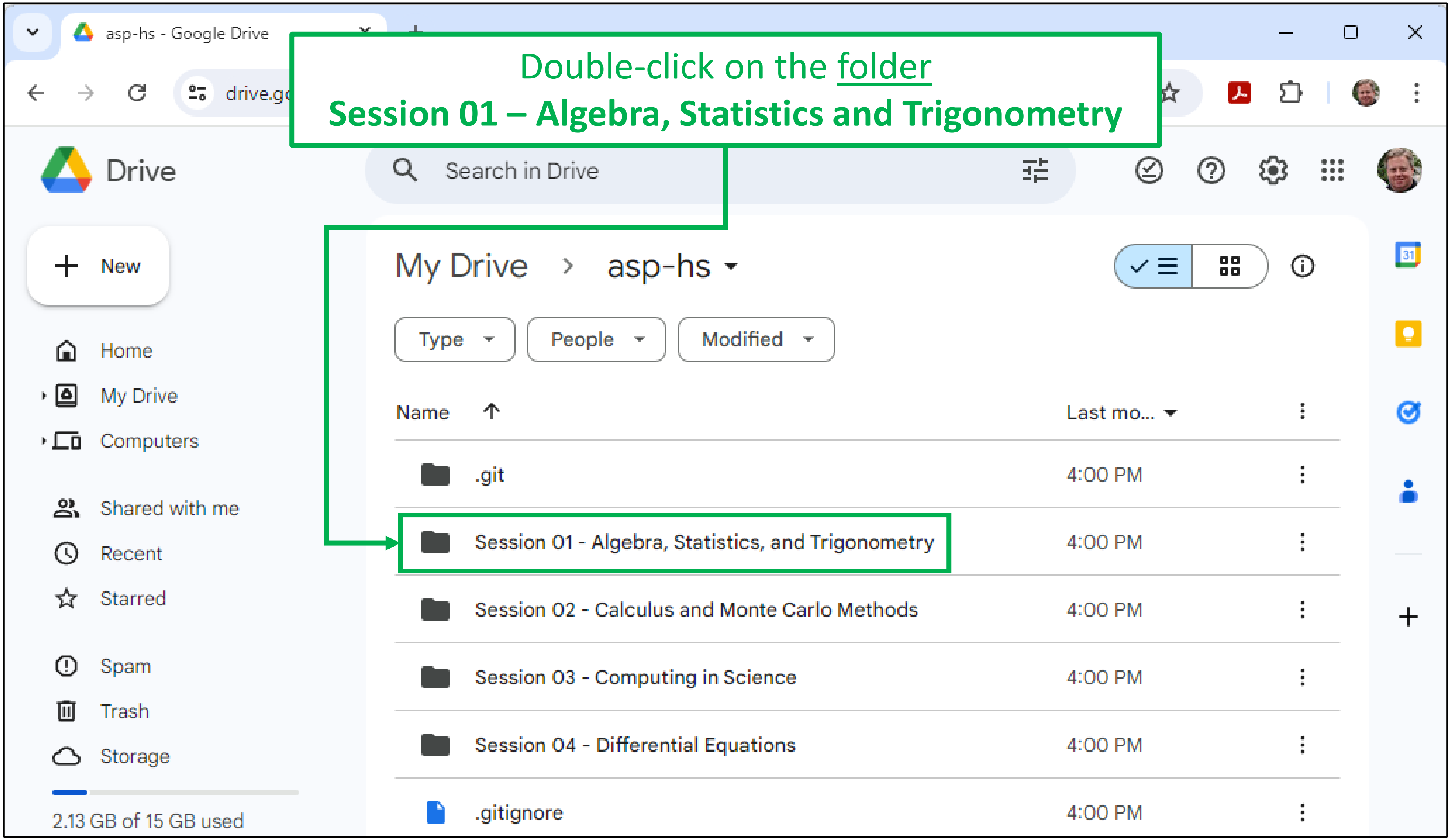


Return to your Google Drive:
https://drive.google.com/drive/my-drive

You should now see a folder called
asp-hs

Double-click that folder to open it

Get the Courseware – **Step 8**



Double-click on the folder

Session 01 – Algebra, Statistics and Trigonometry

My Drive > asp-hs

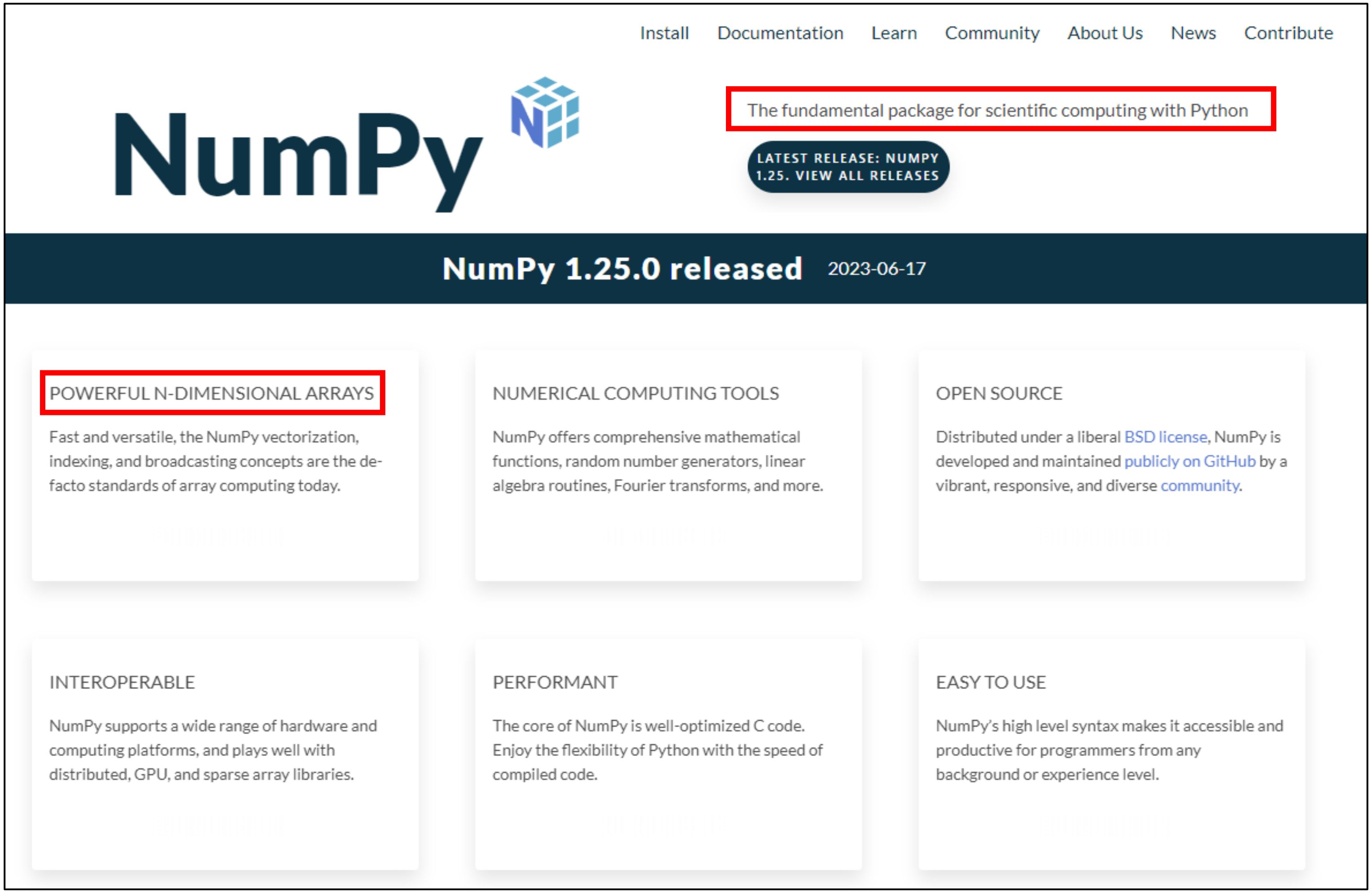
Name	Last mo...
.git	4:00 PM
Session 01 - Algebra, Statistics, and Trigonometry	4:00 PM
Session 02 - Calculus and Monte Carlo Methods	4:00 PM
Session 03 - Computing in Science	4:00 PM
Session 04 - Differential Equations	4:00 PM
.gitignore	4:00 PM

Session **01** – Topics

- Create numerical **arrays** and plot **polynomials**
- Estimate and plot **infinite series** to visualize **convergence**
- Calculate Euclid's **GCD** (HCF) of pairs of random integers
- Calculate the 2nd central moment of **uniform distributions**
- Demonstrate **Euler's Identity** for Complex Numbers
- Use **Polar Coordinates** to draw parametric curves and 2D **random walks**
- Plot the **superposition** of two waves to create *traveling* and standing waves
- Use trigonometry to draw a 3D **sphere** and **torus**

Extending Python via the **numpy** Package

https://numpy.org



The fundamental package for scientific computing with Python

NumPy 1.25.0 released 2023-06-17

POWERFUL N-DIMENSIONAL ARRAYS

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

NUMERICAL COMPUTING TOOLS

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

OPEN SOURCE

Distributed under a liberal BSD license, NumPy is developed and maintained publicly on GitHub by a vibrant, responsive, and diverse community.

INTEROPERABLE

NumPy supports a wide range of hardware and computing platforms, and plays well with distributed, GPU, and sparse array libraries.

PERFORMANT

The core of NumPy is well-optimized C code. Enjoy the flexibility of Python with the speed of compiled code.

EASY TO USE

NumPy's high level syntax makes it accessible and productive for programmers from any background or experience level.

Numpy Arrays

- An **array** is a set of *elements* having all the same **type**
- An individual element in an array is accessed by using its **index number** within square **[]** brackets
  - Every element has a unique index number
  - No two elements share the exact same index number
  - **The first element has an index = 0**
- The function **size()** returns the *length* of an array, which is the number of elements in the array
- The *last* element in an array at **[**size() – 1**]**

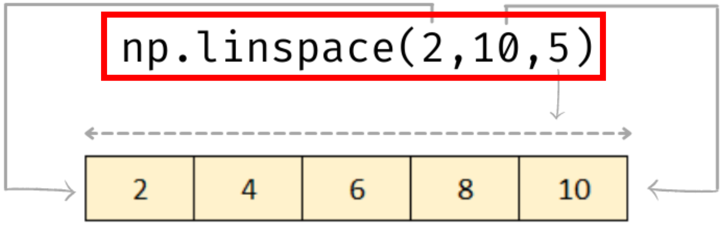
Index Number versus Element **Value**

Index = 2

array_name[2]

A Numpy **Linearly Spaced** Array

Creates a "street" of *mailboxes* where the **values** inside are <u>equally</u> spaced between [start, stop]

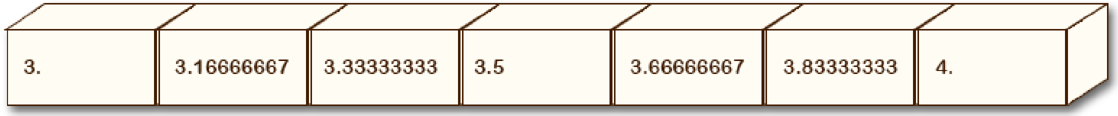


Equally spaced values

```
x = np.linspace(3, 4, 7)
```

Index [0]

Index [6]



np.linspace() figures out the *step* size based on the range of the linear space and the <u>number</u> of elements you request

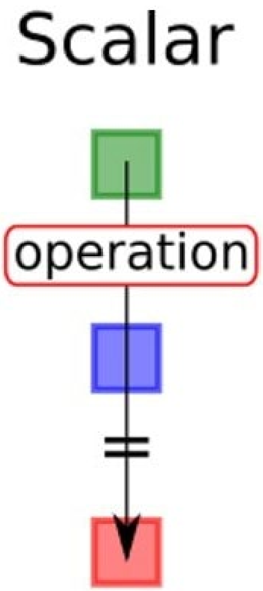
Numpy **Vectorized** Operations

3

*

5

15



a

*

b

a[n]*b[n]

A **vectorized** scalar operation applies a function to every element in a *single* array (to each individual cell)

A **vectorized** array operation applies a function to elements in *both* arrays that have the same <u>index</u> value

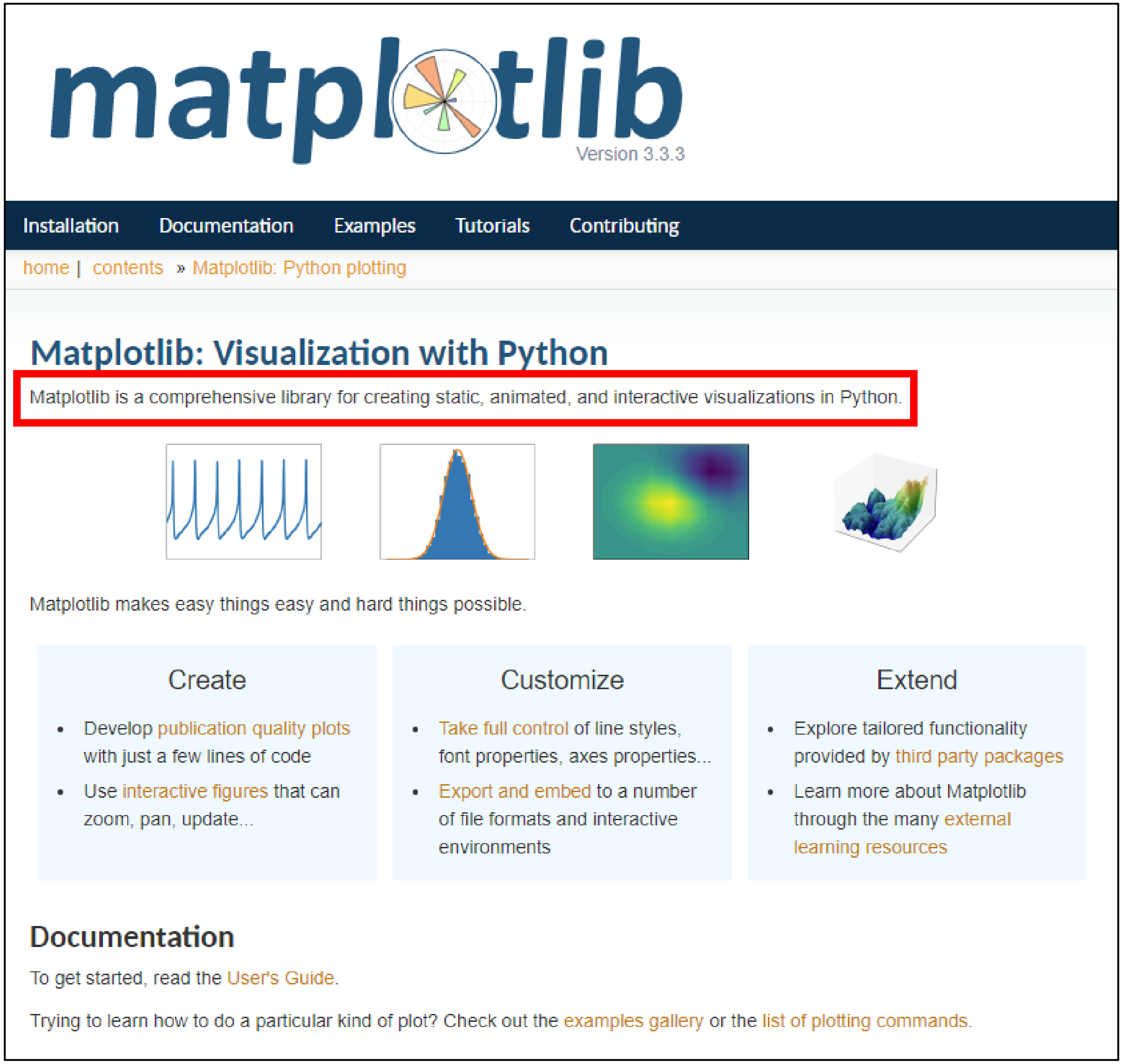
Line Graphs using matplotlib

- Your scientist has asked you to plot the following two functions:

$$y_1 = 2x - 5$$
$$y_2 = -0.3x^2 + 15$$

- The domain for both functions is $-10 \leq x \leq 10$
- You should plot both curves on the same graph

https://matplotlib.org



Matplotlib: Visualization with Python

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.

Matplotlib makes easy things easy and hard things possible.

Create

- Develop publication quality plots with just a few lines of code
- Use interactive figures that can zoom, pan, update...

Customize

- Take full control of line styles, font properties, axes properties...
- Export and embed to a number of file formats and interactive environments

Extend

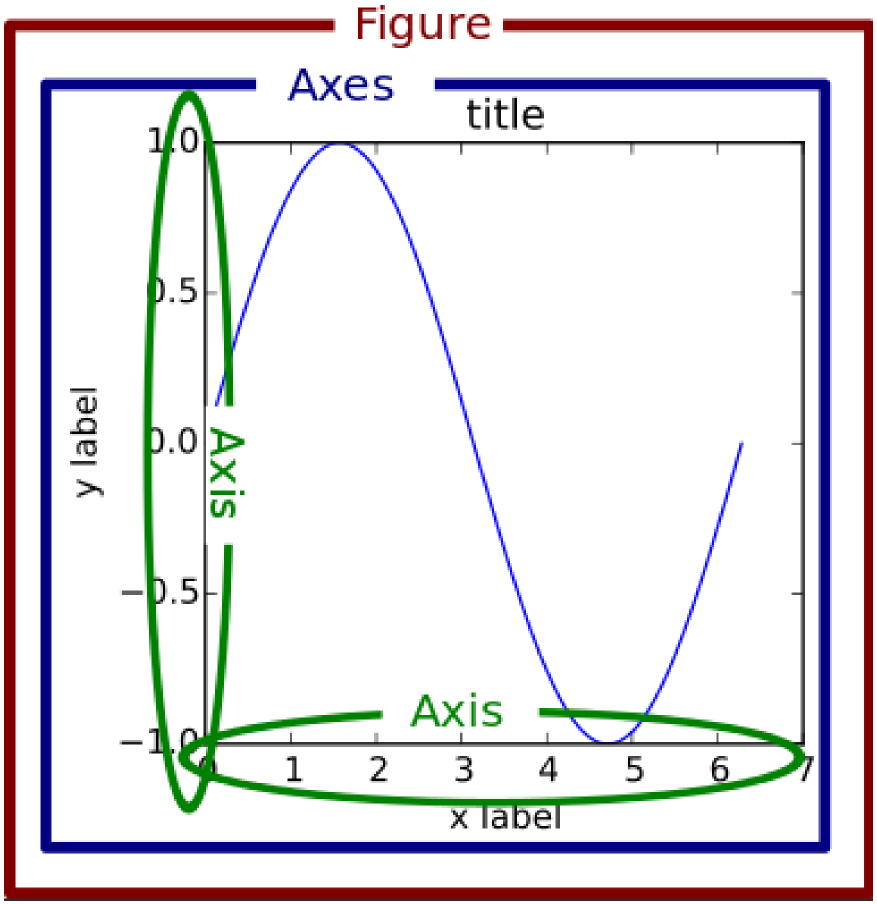
- Explore tailored functionality provided by third party packages
- Learn more about Matplotlib through the many external learning resources

Documentation

To get started, read the User's Guide.

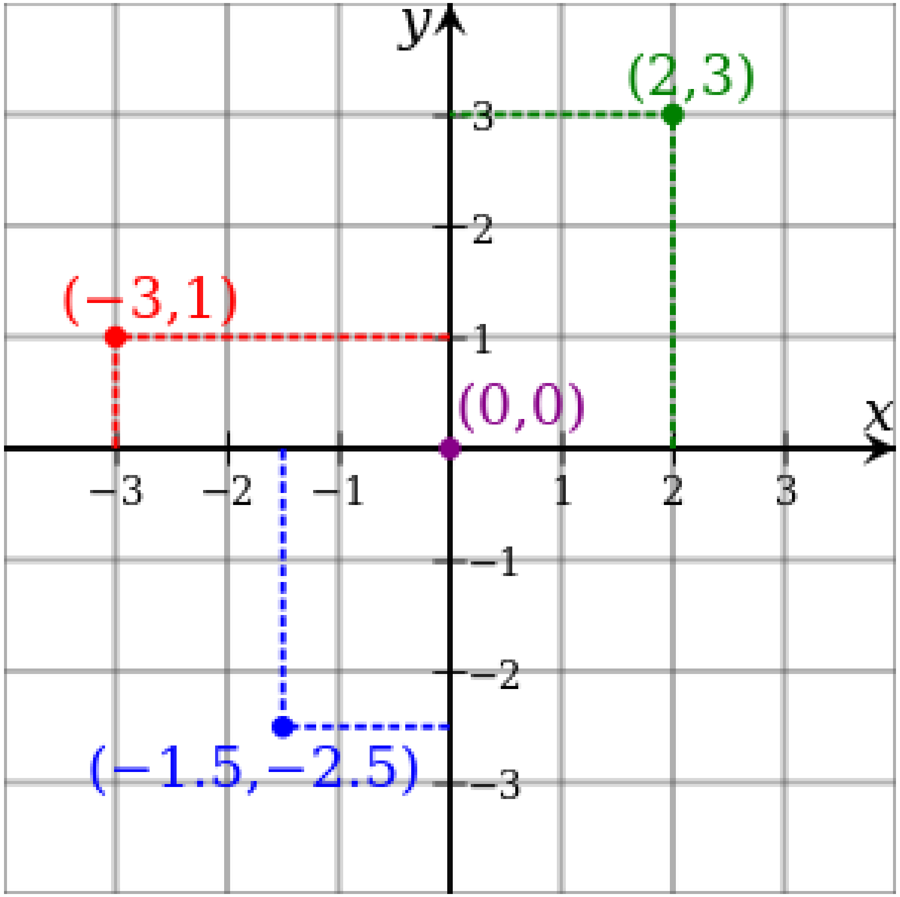
Trying to learn how to do a particular kind of plot? Check out the examples gallery or the list of plotting commands.

Matplotlib Container Hierarchy

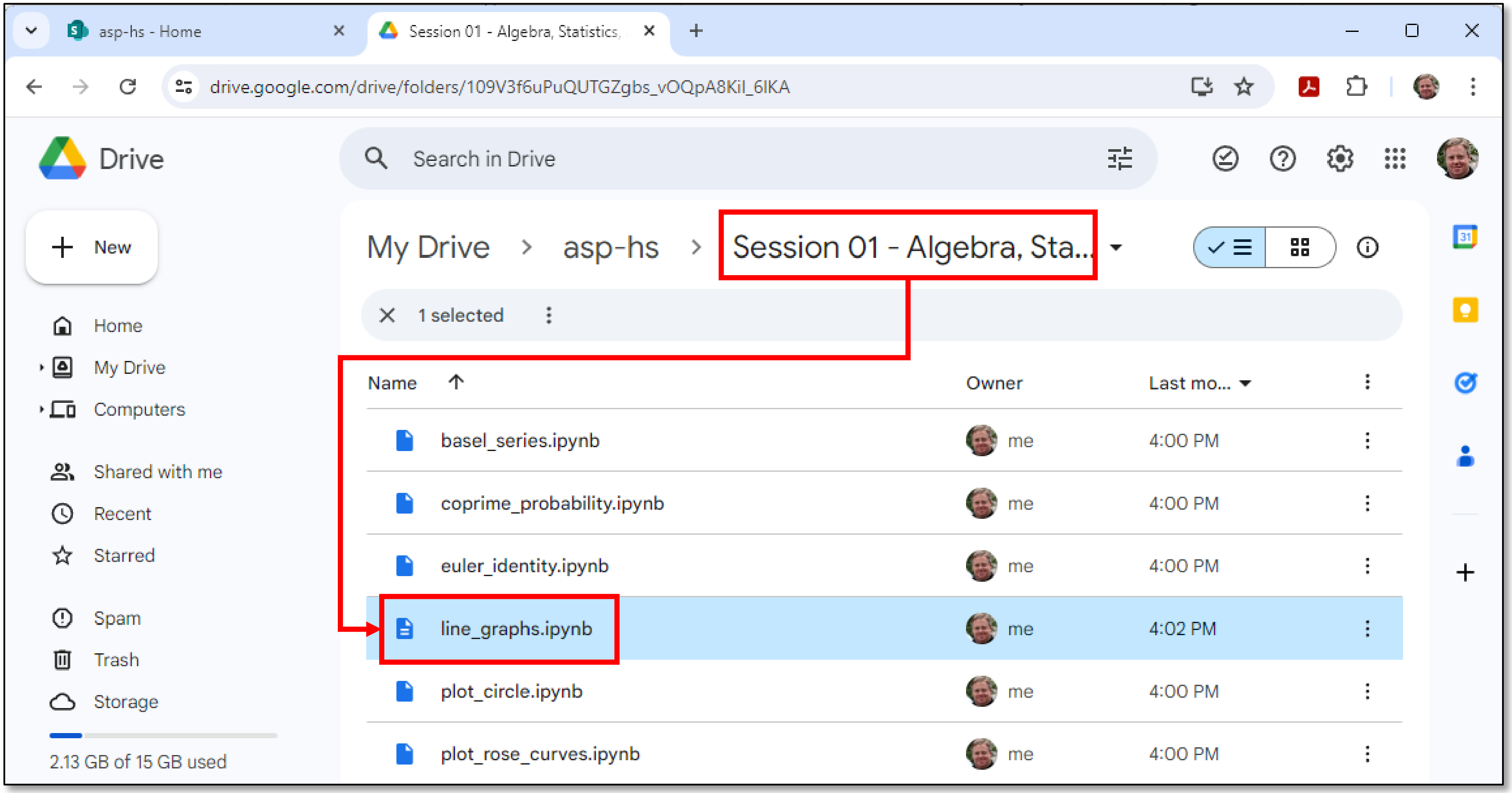


Cartesian Coordinates

Created by
René Descartes
in 1637



Open line_graphs.ipynb

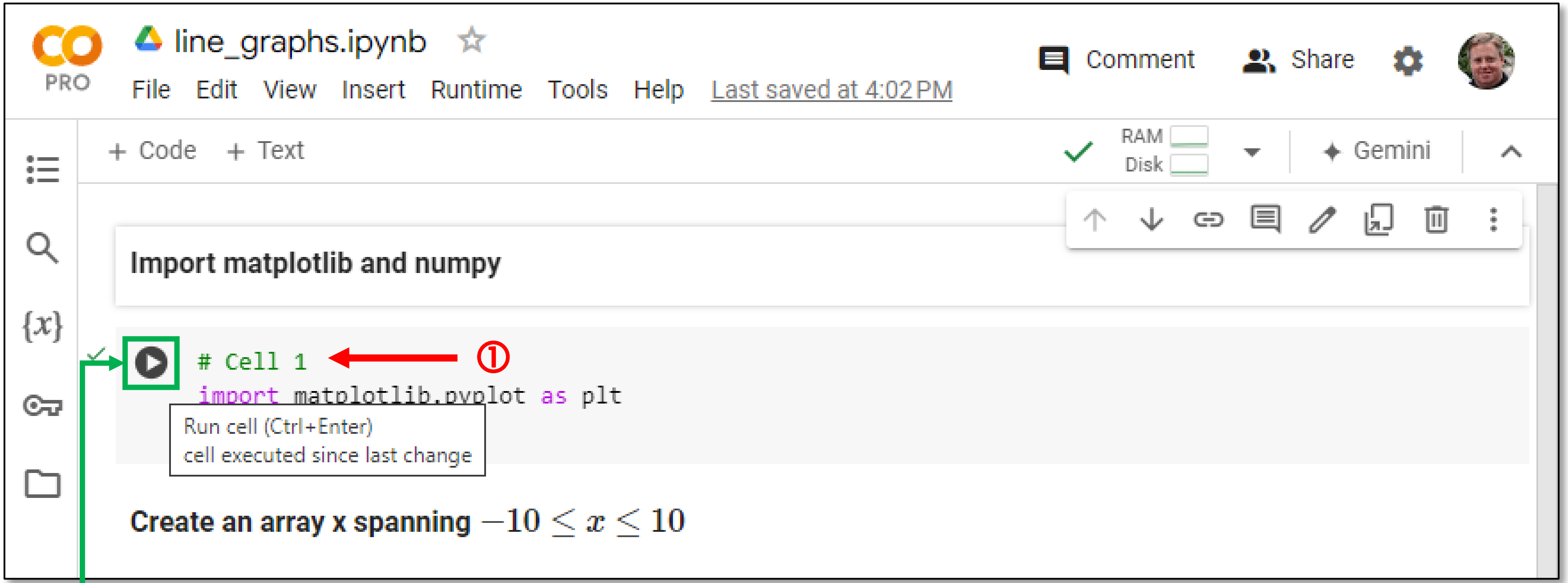


asp-hs - Home
Session 01 - Algebra, Statistics,
drive.google.com/drive/folders/109V3f6uPuQUTGZgbs_vOQpA8Kil_6IKA
Drive
Search in Drive
New
Home
My Drive
Computers
Shared with me
Recent
Starred
Spam
Trash
Storage
2.13 GB of 15 GB used
My Drive › asp-hs › Session 01 - Algebra, Sta...
1 selected

Name ↑	Owner	Last mo...
basel_series.ipynb	me	4:00 PM
coprime_probability.ipynb	me	4:00 PM
euler_identity.ipynb	me	4:00 PM
line_graphs.ipynb	me	4:02 PM
plot_circle.ipynb	me	4:00 PM
plot_rose_curves.ipynb	me	4:00 PM

Double-click on a notebook to open it

Run line_graphs.ipynb – **Cell 1**


CO PRO
line_graphs.ipynb
File Edit View Insert Runtime Tools Help Last saved at 4:02 PM
Comment
Share
+ Code + Text
RAM
Disk
Gemini
①
Run cell (Ctrl+Enter)
cell executed since last change

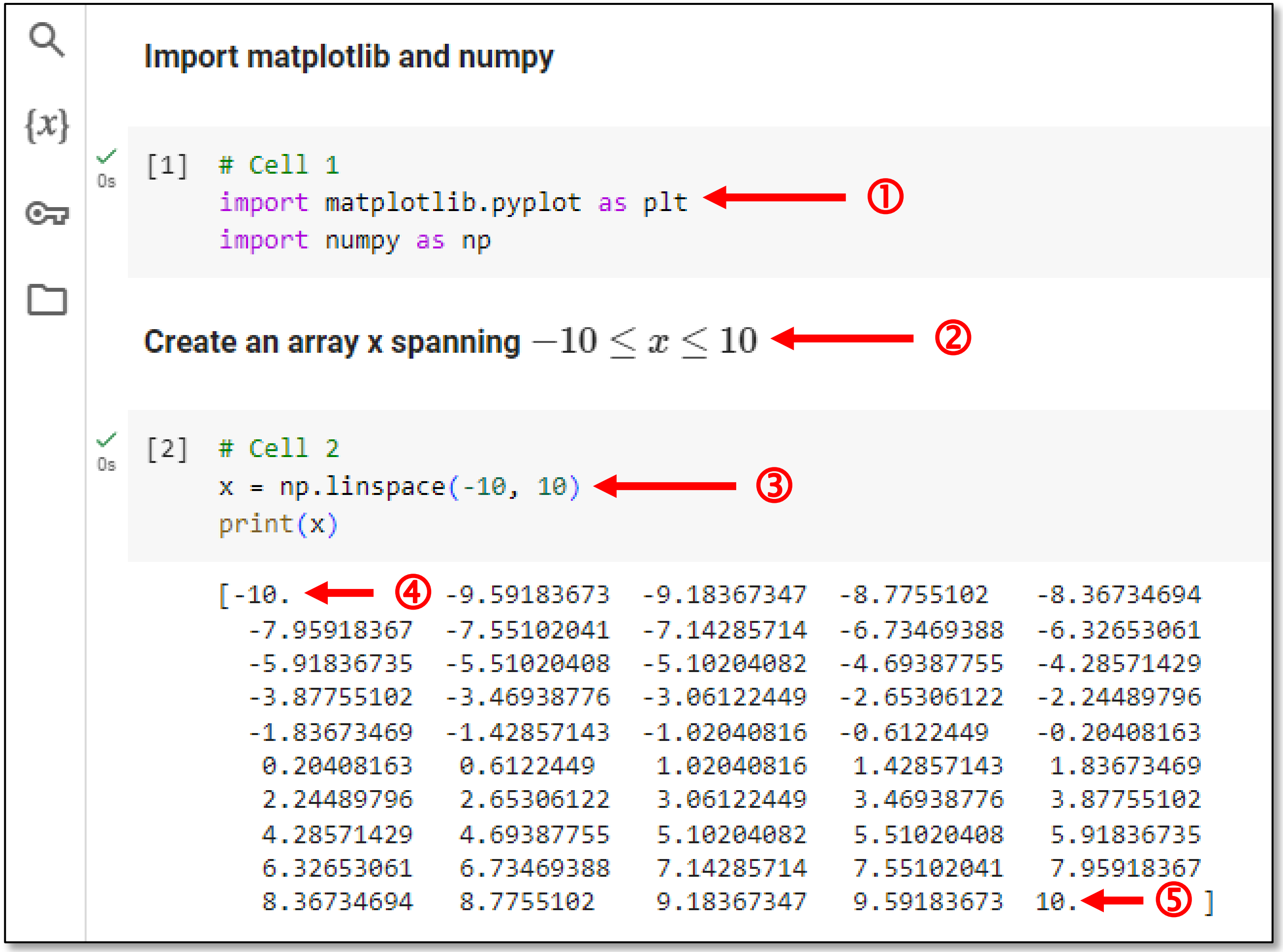
Import matplotlib and numpy

```
# Cell 1
import matplotlib.pyplot as plt
```

Create an array x spanning $-10 \leq x \leq 10$

Click the "Run" (play) button to execute this cell in the notebook

Run line_graphs.ipynb – Cells 1...2



Import matplotlib and numpy

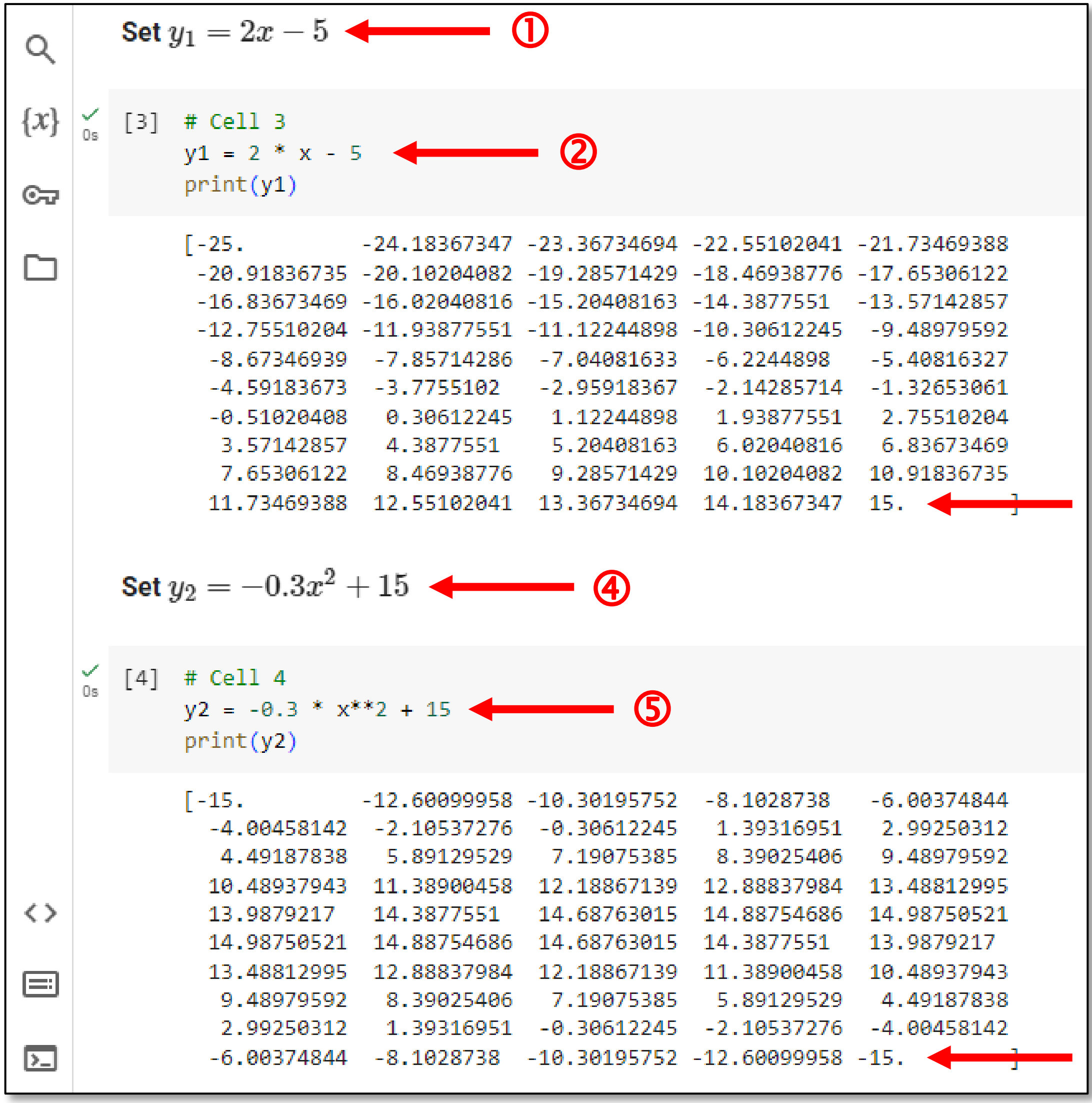
```
[1]  # Cell 1
     import matplotlib.pyplot as plt  ①
     import numpy as np
```

Create an array x spanning $-10 \leq x \leq 10$ ②

```
[2]  # Cell 2
     x = np.linspace(-10, 10)  ③
     print(x)
```

```
[-10.  ④   -9.59183673 -9.18367347 -8.7755102  -8.36734694
  -7.95918367 -7.55102041 -7.14285714 -6.73469388 -6.32653061
  -5.91836735 -5.51020408 -5.10204082 -4.69387755 -4.28571429
  -3.87755102 -3.46938776 -3.06122449 -2.65306122 -2.24489796
  -1.83673469 -1.42857143 -1.02040816 -0.6122449  -0.20408163
   0.20408163  0.6122449   1.02040816  1.42857143  1.83673469
   2.24489796  2.65306122  3.06122449  3.46938776  3.87755102
   4.28571429  4.69387755  5.10204082  5.51020408  5.91836735
   6.32653061  6.73469388  7.14285714  7.55102041  7.95918367
   8.36734694  8.7755102   9.18367347  9.59183673 10.  ⑤ ]
```

Run line_graphs.ipynb – Cells 3…4



Set $y_1 = 2x - 5$ ①

```
[3] # Cell 3
    y1 = 2 * x - 5
    print(y1)
```

②

```
[-25.         -24.18367347 -23.36734694 -22.55102041 -21.73469388
 -20.91836735 -20.10204082 -19.28571429 -18.46938776 -17.65306122
 -16.83673469 -16.02040816 -15.20408163 -14.3877551  -13.57142857
 -12.75510204 -11.93877551 -11.12244898 -10.30612245  -9.48979592
  -8.67346939  -7.85714286  -7.04081633  -6.2244898   -5.40816327
  -4.59183673  -3.7755102   -2.95918367  -2.14285714  -1.32653061
  -0.51020408   0.30612245   1.12244898   1.93877551   2.75510204
   3.57142857   4.3877551    5.20408163   6.02040816   6.83673469
   7.65306122   8.46938776   9.28571429  10.10204082  10.91836735
  11.73469388  12.55102041  13.36734694  14.18367347  15.        ]
```

③

$$= 2(10) - 5$$

$$= 20 - 5$$

$$= 15$$

Set $y_2 = -0.3x^2 + 15$ ④

```
[4] # Cell 4
    y2 = -0.3 * x**2 + 15
    print(y2)
```

⑤

```
[-15.         -12.60099958 -10.30195752  -8.1028738   -6.00374844
  -4.00458142  -2.10537276  -0.30612245   1.39316951   2.99250312
   4.49187838   5.89129529   7.19075385   8.39025406   9.48979592
  10.48937943  11.38900458  12.18867139  12.88837984  13.48812995
  13.9879217   14.3877551   14.68763015  14.88754686  14.98750521
  14.98750521  14.88754686  14.68763015  14.3877551   13.9879217
  13.48812995  12.88837984  12.18867139  11.38900458  10.48937943
   9.48979592   8.39025406   7.19075385   5.89129529   4.49187838
   2.99250312   1.39316951  -0.30612245  -2.10537276  -4.00458142
  -6.00374844  -8.1028738  -10.30195752 -12.60099958 -15.        ]
```

⑥

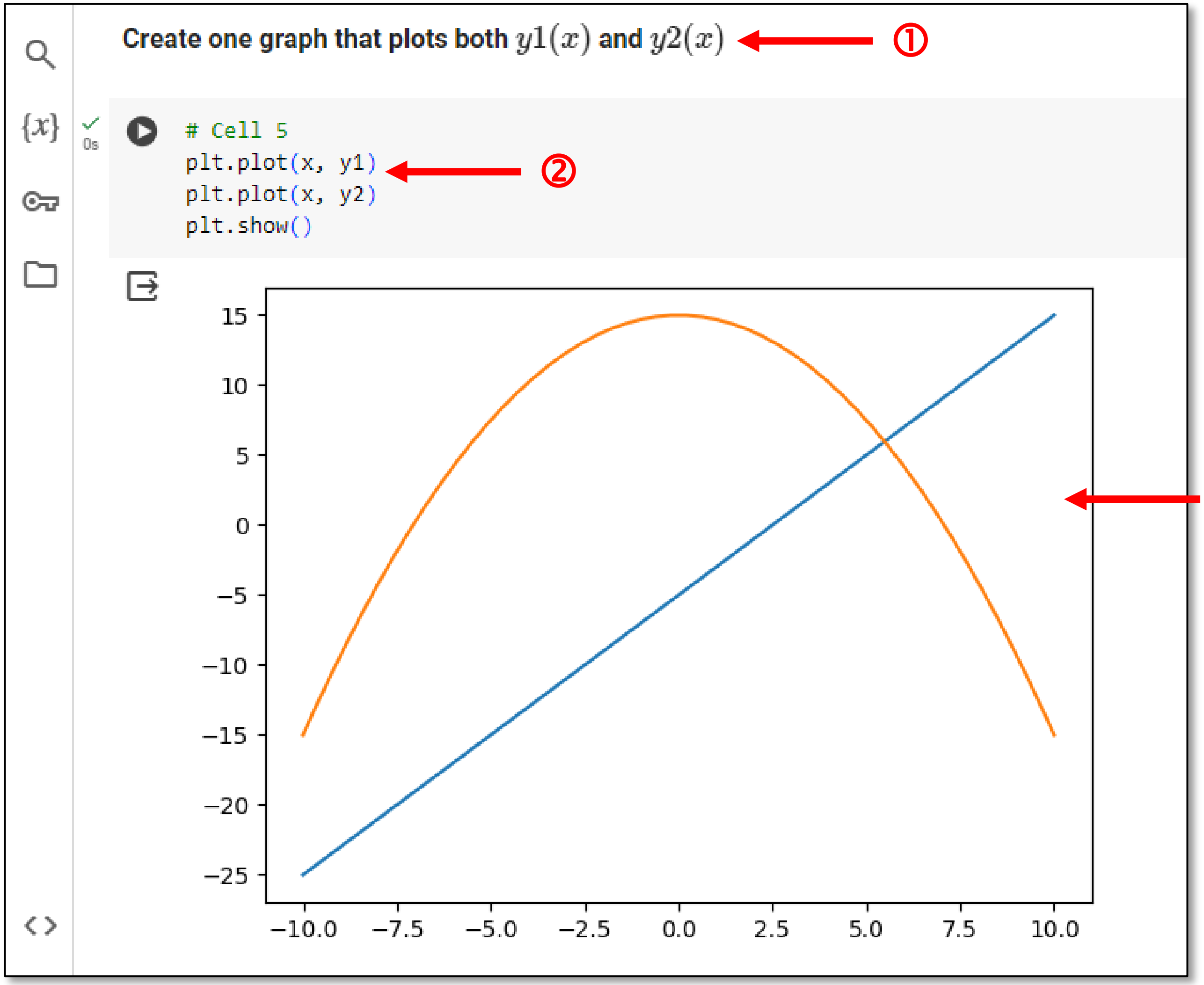
$$= -0.3(10^2) + 15$$

$$= -0.3(100) + 15$$

$$= -30 + 15$$

$$= -15$$

Run line_graphs.ipynb – Cell 5



Create one graph that plots both $y1(x)$ and $y2(x)$ ①

```
# Cell 5
plt.plot(x, y1)
plt.plot(x, y2)
plt.show()
```

②

③

Infinite Series (Sums)

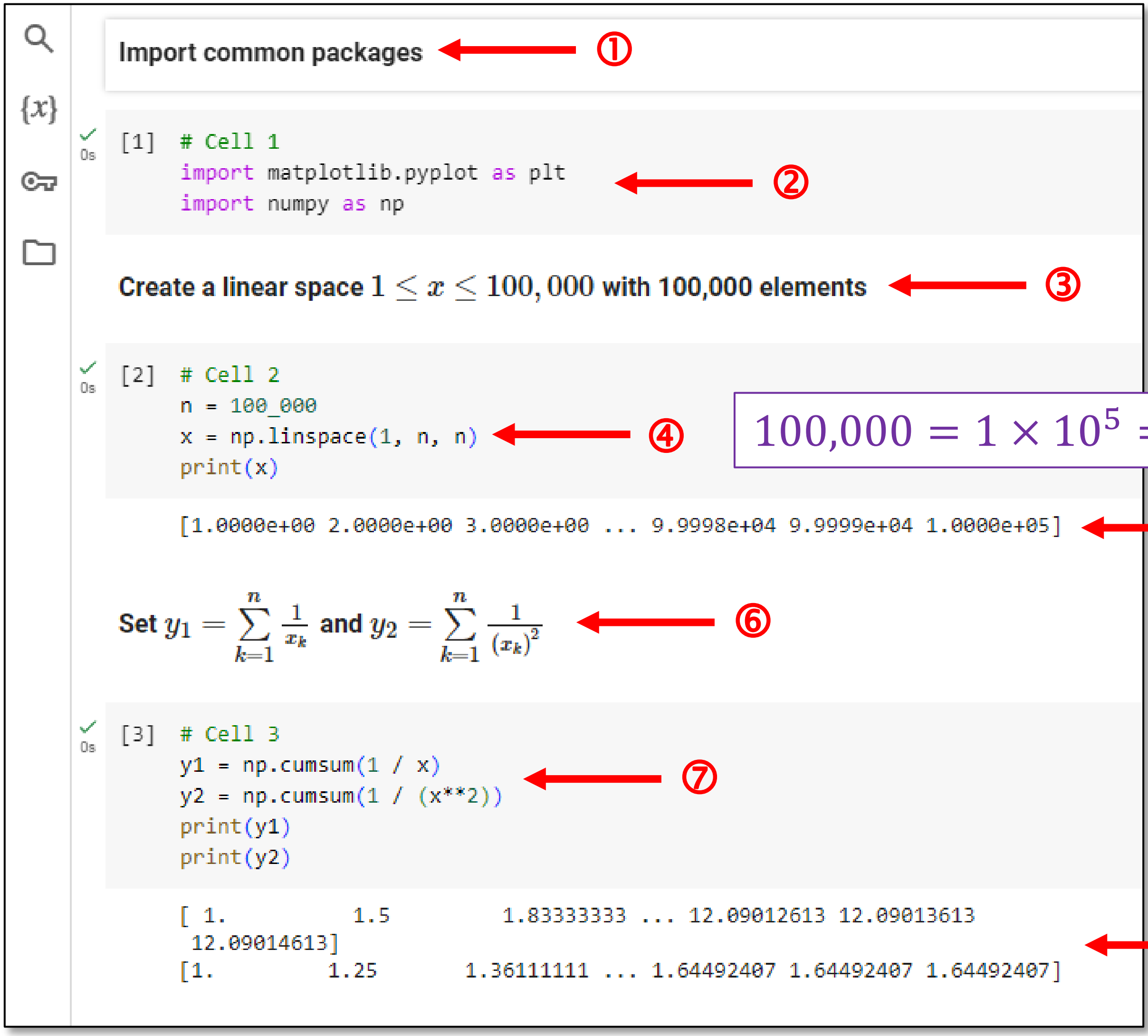
$$y_1 = \sum_{n=1}^{\infty} \frac{1}{n} = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \frac{1}{7} + \cdots$$

- This sum is called the **Harmonic series**
- Does the Harmonic series **converge** to a single value or **diverge** (grow without bounds)?

$$y_2 = \sum_{n=1}^{\infty} \frac{1}{n^2} = 1 + \frac{1}{4} + \frac{1}{9} + \frac{1}{16} + \frac{1}{25} + \frac{1}{36} + \frac{1}{49} + \cdots$$

- This sum is called the **Basel series**
- Find the value of $\sqrt{6y_2}$ when $n = 100{,}000$

Run basel_series.ipynb – Cells **1…3**



Import common packages ①

```
[1]  # Cell 1
     import matplotlib.pyplot as plt
     import numpy as np
```

②

Create a linear space $1 \le x \le 100,000$ with 100,000 elements ③

```
[2]  # Cell 2
     n = 100_000
     x = np.linspace(1, n, n)
     print(x)
```

④

$100,000 = 1 \times 10^5 =$ `1.0e+05`

```
[1.0000e+00 2.0000e+00 3.0000e+00 ... 9.9998e+04 9.9999e+04 1.0000e+05]
```

⑤

Set $y_1 = \sum_{k=1}^{n} \frac{1}{x_k}$ and $y_2 = \sum_{k=1}^{n} \frac{1}{(x_k)^2}$ ⑥

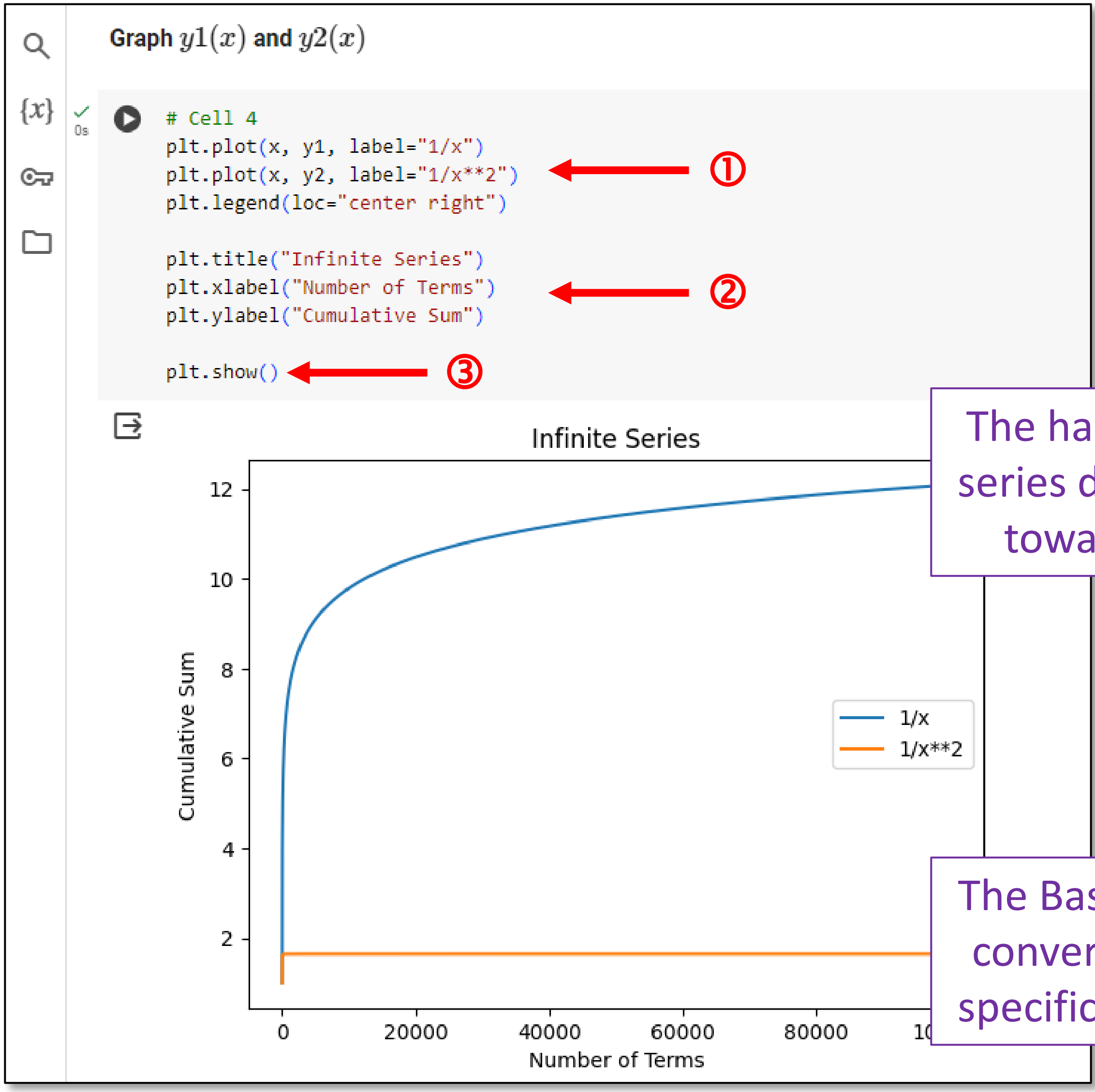
```
[3]  # Cell 3
     y1 = np.cumsum(1 / x)
     y2 = np.cumsum(1 / (x**2))
     print(y1)
     print(y2)
```

⑦

```
[ 1.          1.5         1.83333333 ... 12.09012613 12.09013613
 12.09014613]
[1.         1.25       1.36111111 ... 1.64492407 1.64492407 1.64492407]
```

⑧

Run basel_series.ipynb – Cell 4



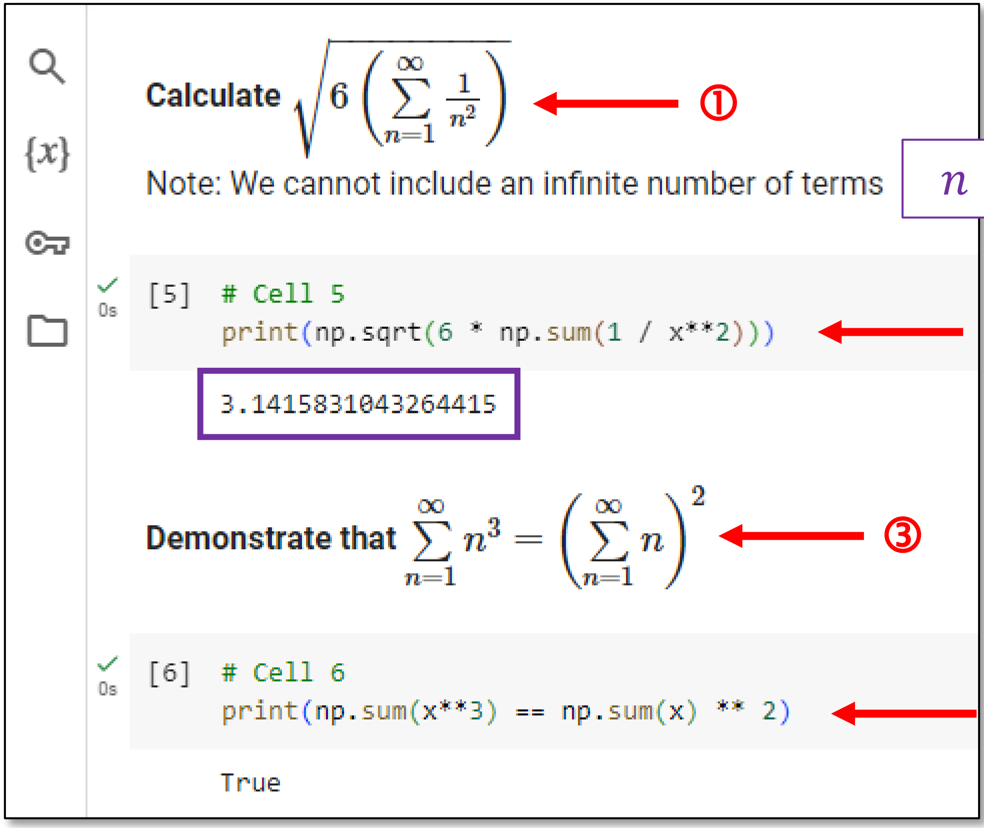
Graph $y1(x)$ and $y2(x)$

```
# Cell 4
plt.plot(x, y1, label="1/x")
plt.plot(x, y2, label="1/x**2")          ①
plt.legend(loc="center right")

plt.title("Infinite Series")
plt.xlabel("Number of Terms")            ②
plt.ylabel("Cumulative Sum")

plt.show()          ③
```

The harmonic series diverges toward ∞

The Basel series converges to a specific number

Run basel_series.ipynb – Cells **5…6**



Calculate $\sqrt{6\left(\sum_{n=1}^{\infty}\frac{1}{n^2}\right)}$ ①

Note: We cannot include an infinite number of terms

$n = 100{,}000$

```
[5] # Cell 5
    print(np.sqrt(6 * np.sum(1 / x**2)))
```
②

```
3.1415831043264415
```

Demonstrate that $\sum_{n=1}^{\infty} n^3 = \left(\sum_{n=1}^{\infty} n\right)^2$ ③

```
[6] # Cell 6
    print(np.sum(x**3) == np.sum(x) ** 2)
```
④

```
True
```

The sum of n **cubed** equals the sum **squared** of n

The Basel Problem

Leonhard Euler
(1707 – 1783)

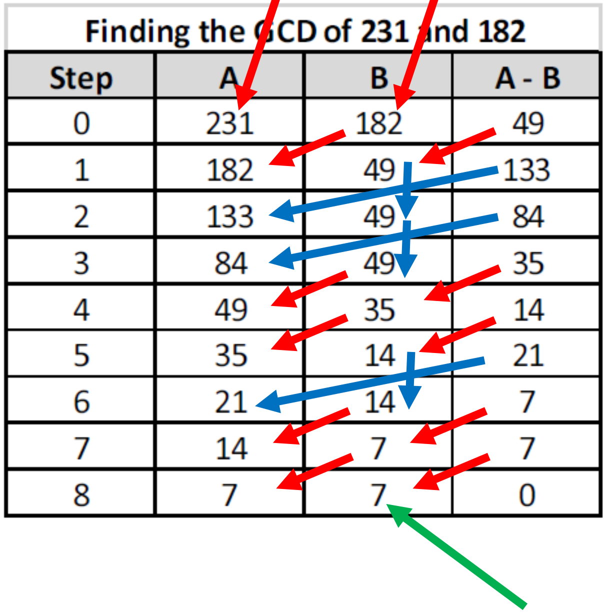
$$\boxed{\sum_{k=1}^{\infty} \frac{1}{k^2} = \frac{\pi^2}{6}}$$

288 years later, we still do not know the exact value of

$$\sum_{n=1}^{\infty} \frac{1}{n^3}$$

Greatest Common Divisor (GCD)

Example: **What is the GCD of 231 and 182?** In step 0, **A** is always greater than or equal to **B.** In steps 1 and beyond, the **A** value is the *greater* of the prior step's B or (A-B) values. The **B** value is the *lesser* of either the prior step's B or (A – B) values. The algorithm stops when A – B = 0, and the GCD was the very last **B** value. Follow along with each step in the table below:



Finding the GCD of 231 and 182

Step	A	B	A - B
0	231	182	49
1	182	49	133
2	133	49	84
3	84	49	35
4	49	35	14
5	35	14	21
6	21	14	7
7	14	7	7
8	7	7	0

What divides A and B must also divide the ***difference*** of A - B

Why?

Given {A, B, a, b, r} ∈ ℤ

$$A = a * r, B = b * r$$

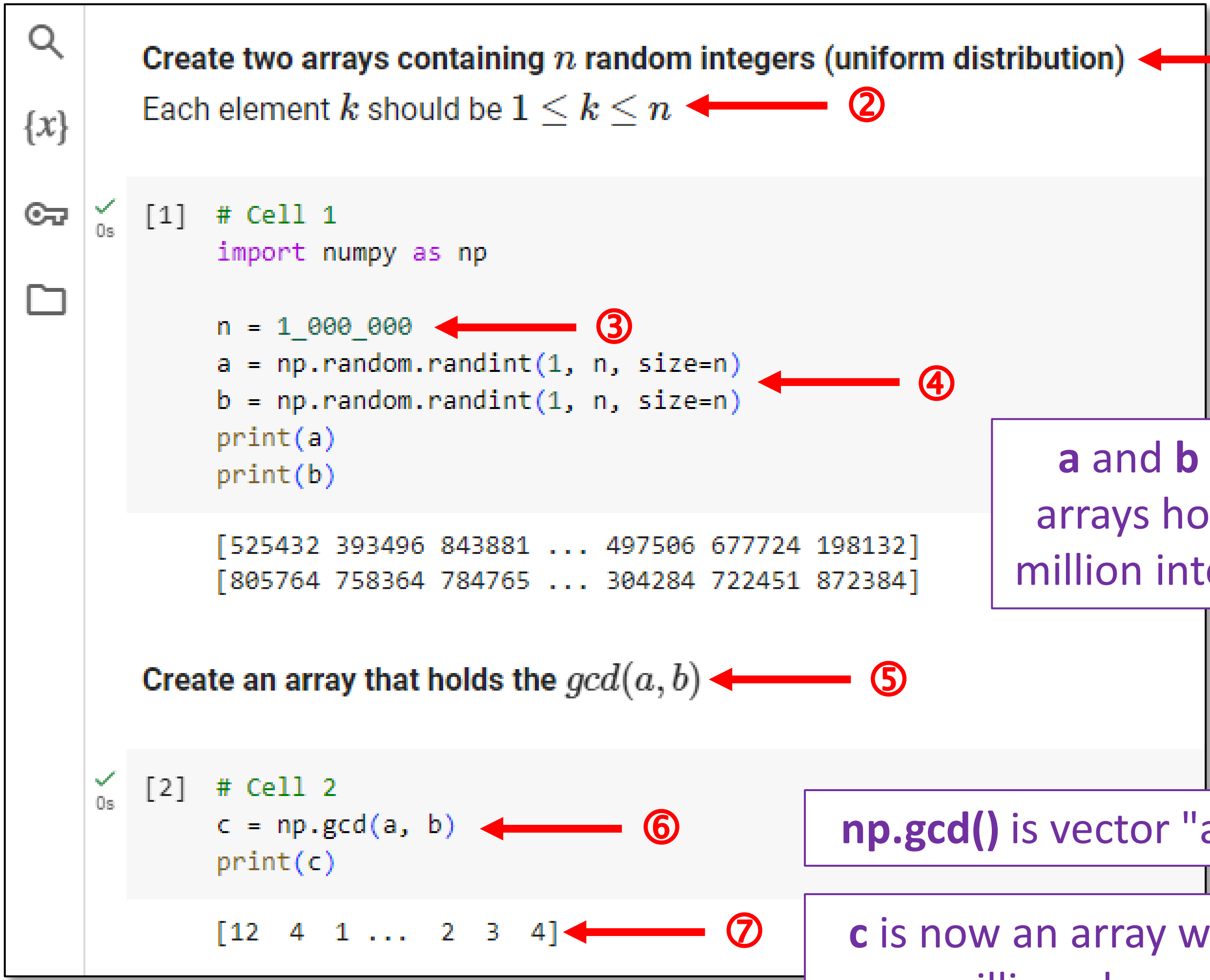
$$(A - B) = a * r - b * r$$

$$a - b = \frac{(A - B)}{r}$$

Coprime Probability

- Your scientist needs you to write a program to *estimate* the probability $\boldsymbol{p}$ that any two positive random integers are **coprime**
- Two numbers are **coprime** if they share **no common factors**
- For example, the numbers 6 and 35 are **not prime** because $6 = 2 \times 3$ and $35 = 5 \times 7$
- However, when **compared to each other**, 6 and 35 are **coprime** because they share **no common factors**
- She wants you to sample **one million pairs** of random integers between one and one million inclusive
- She wants to know the value of $\sqrt{\frac{6}{p}}$

Run coprime_probability.ipynb – Cells 1…2



Create two arrays containing n random integers (uniform distribution) ①

Each element k should be $1 \leq k \leq n$ ②

```
[1]  # Cell 1
     import numpy as np

     n = 1_000_000
     a = np.random.randint(1, n, size=n)
     b = np.random.randint(1, n, size=n)
     print(a)
     print(b)
```

③ ④

```
[525432 393496 843881 ... 497506 677724 198132]
[805764 758364 784765 ... 304284 722451 872384]
```

a and **b** are now arrays holding one million integers each

Create an array that holds the $gcd(a, b)$ ⑤

```
[2]  # Cell 2
     c = np.gcd(a, b)
     print(c)
```

⑥

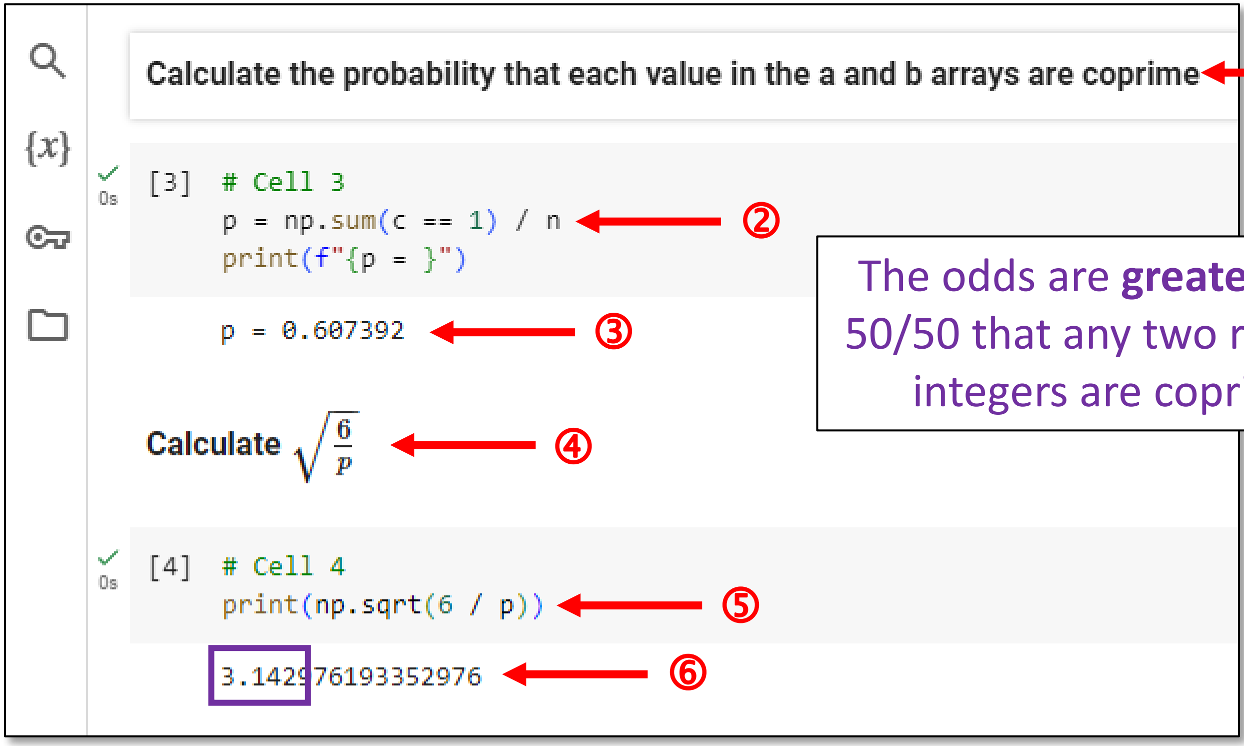
```
[12  4  1 ...  2  3  4]
```

⑦

np.gcd() is vector "aware"

c is now an array with one million elements

Run coprime_probability.ipynb – **Cells 3…4**



Calculate the probability that each value in the a and b arrays are coprime

```
[3]  # Cell 3
     p = np.sum(c == 1) / n
     print(f"{p = }")
```

```
p = 0.607392
```

Calculate $\sqrt{\frac{6}{p}}$

```
[4]  # Cell 4
     print(np.sqrt(6 / p))
```

```
3.142976193352976
```

The odds are **greater than** 50/50 that any two random integers are coprime

If GCD(a, b) == 1 then a and b are coprime

Probability is the number of times something **did happen** *divided* by the number of times it **could have happened**

Computing with Random Numbers?

$$0.607927102 \approx 61\% \approx \boxed{\frac{6}{\pi^2}}$$

$$\frac{1}{1^2}+\frac{1}{2^2}+\frac{1}{3^2}+\frac{1}{4^2}+\cdots=\boxed{\frac{\pi^2}{6}}$$

Leonhard Euler
(1707-1783)

Euler **noticed things** that many others did not…

Variance of Uniform Distributions

- Your scientist needs a program that can:
  - Generate 15 sets of **random sizes** between **10,000** and **200,000** items
  - Within each set, every item is a random integer chosen within a range between a random **lower limit** and a random **upper limit**
  - The **lower limit** for each set is a random number between **0 and 10,000**
  - The **upper limit** is that set's lower limit **plus** another random number between 0 and 100,000
  - Calculate the mean ($\boldsymbol{\mu}$) and variance ($\boldsymbol{\sigma}^2$) for each set's population

$$\boxed{\sigma^2 = \frac{1}{n}\sum_{i=i}^{n}(x_i - \mu)^2 \;\; where \;\; \mu = \frac{1}{n}\sum_{i=1}^{n} x_i}$$

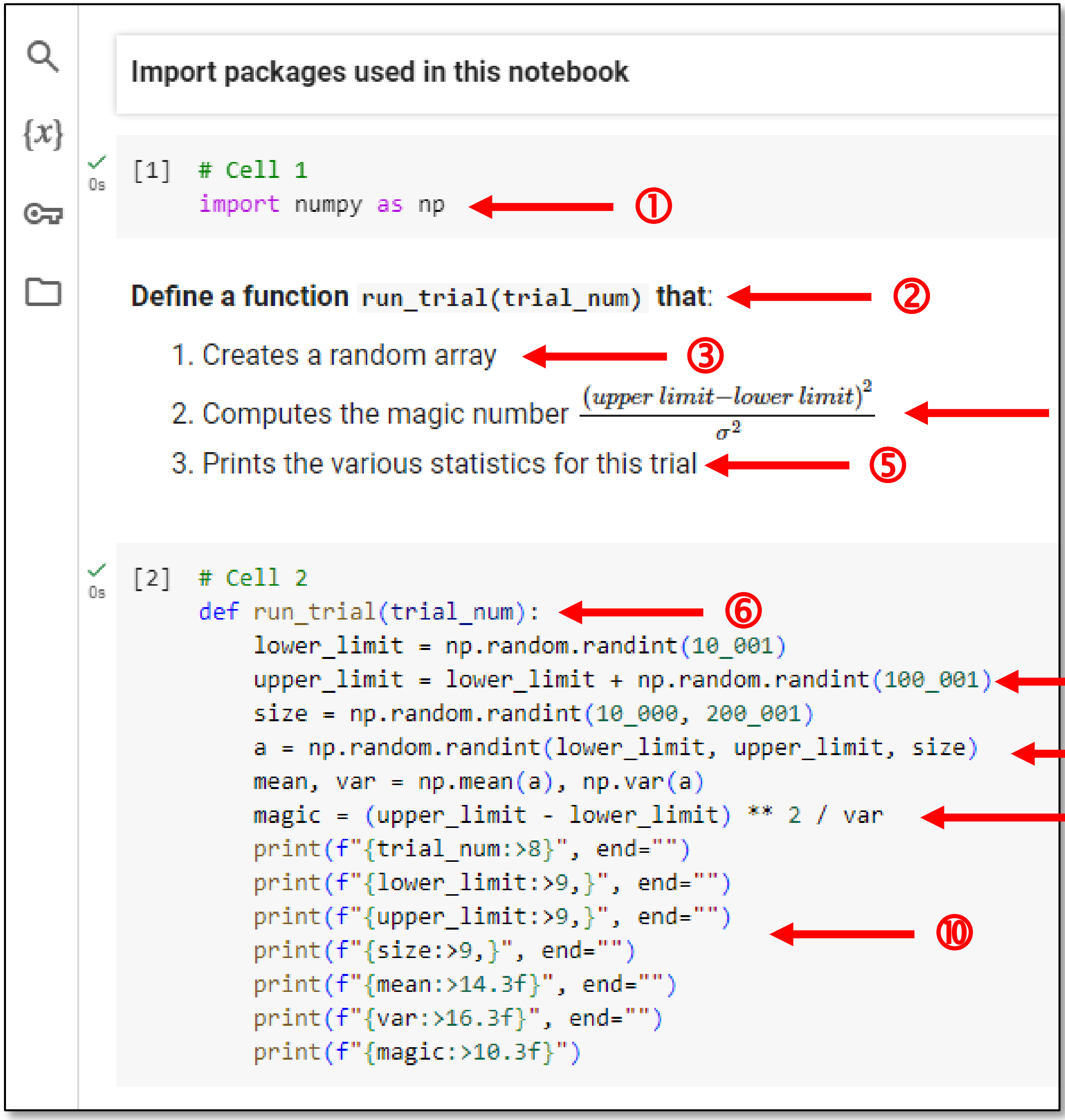
Variance of Uniform Distributions

- The research goal is to determine if a magic number hides within **all** *uniform* random number distributions
  - Calculate and display this “constant” for each set:

$$\textit{Magic Number} = \frac{(\boldsymbol{upperLimit} - \boldsymbol{lowerLimit})^{\mathbf{2}}}{variance}$$

  - Is this number the same for ALL uniform distributions?
- Can we use this value to test if dice are loaded?

Run uniform_variance.ipynb – Cells 1…2



Import packages used in this notebook

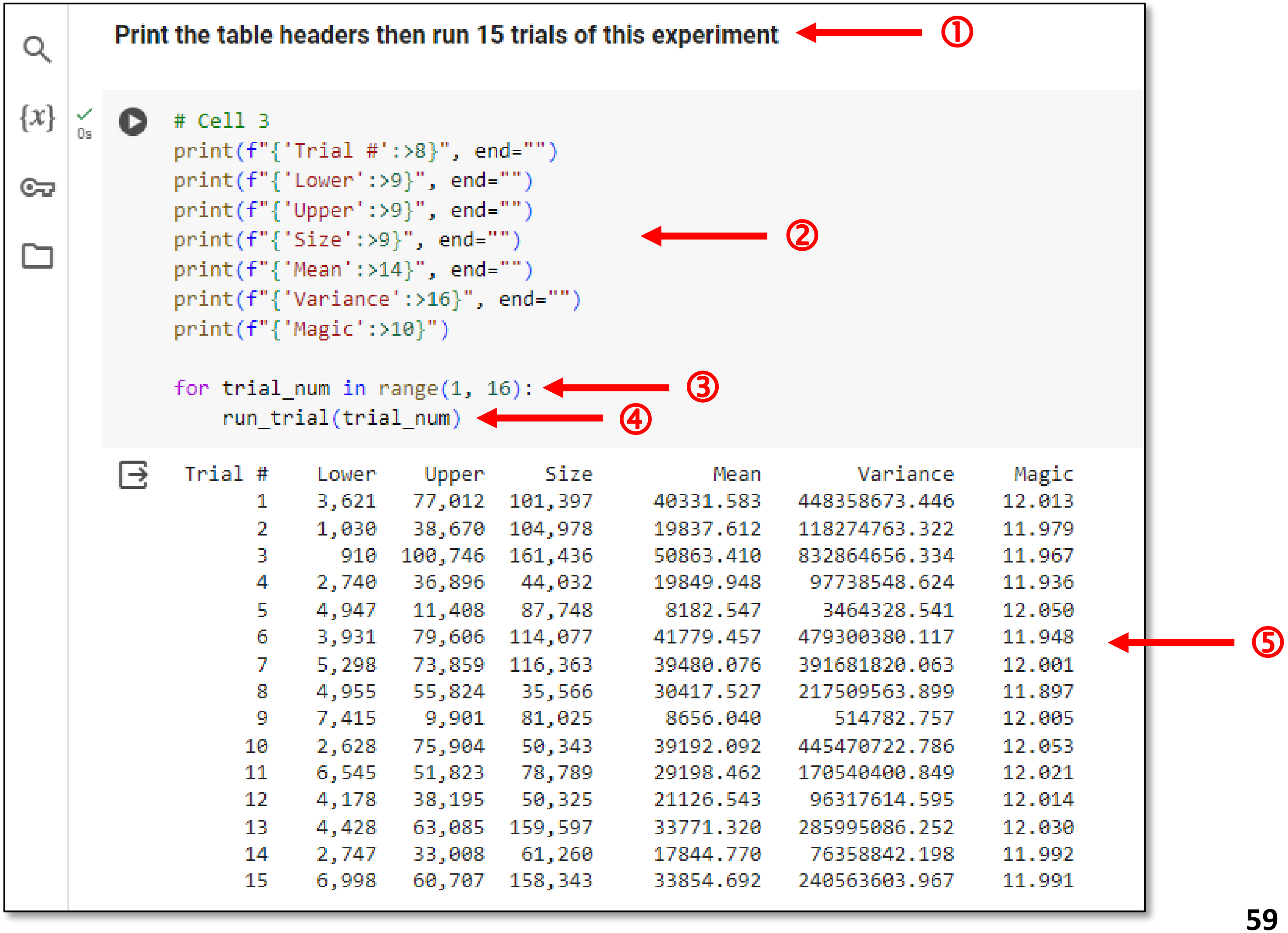
```
[1]  # Cell 1
     import numpy as np
```

Define a function `run_trial(trial_num)` that:

1. Creates a random array
2. Computes the magic number $\frac{(upper\ limit - lower\ limit)^2}{\sigma^2}$
3. Prints the various statistics for this trial

```
[2]  # Cell 2
     def run_trial(trial_num):
         lower_limit = np.random.randint(10_001)
         upper_limit = lower_limit + np.random.randint(100_001)
         size = np.random.randint(10_000, 200_001)
         a = np.random.randint(lower_limit, upper_limit, size)
         mean, var = np.mean(a), np.var(a)
         magic = (upper_limit - lower_limit) ** 2 / var
         print(f"{trial_num:>8}", end="")
         print(f"{lower_limit:>9,}", end="")
         print(f"{upper_limit:>9,}", end="")
         print(f"{size:>9,}", end="")
         print(f"{mean:>14.3f}", end="")
         print(f"{var:>16.3f}", end="")
         print(f"{magic:>10.3f}")
```

Run uniform_variance.ipynb – **Cell 3**

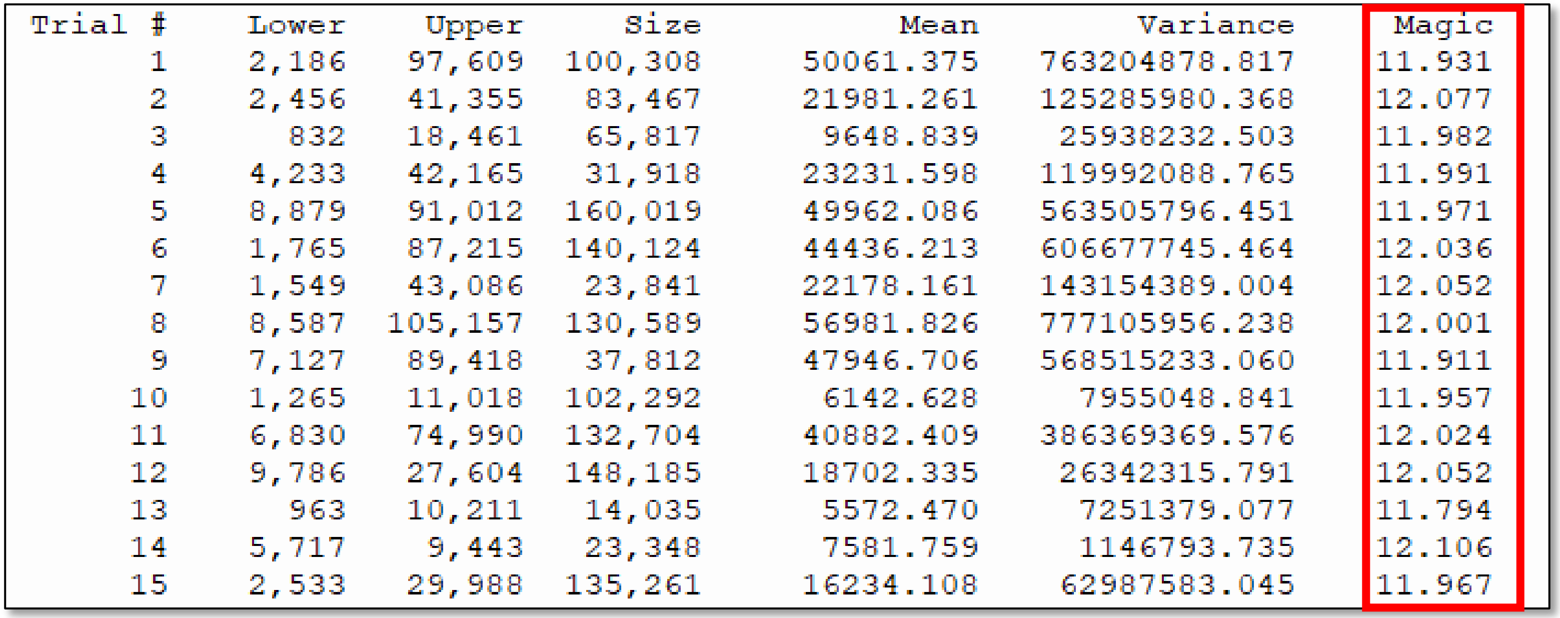


Print the table headers then run 15 trials of this experiment

```
# Cell 3
print(f"{'Trial #':>8}", end="")
print(f"{'Lower':>9}", end="")
print(f"{'Upper':>9}", end="")
print(f"{'Size':>9}", end="")
print(f"{'Mean':>14}", end="")
print(f"{'Variance':>16}", end="")
print(f"{'Magic':>10}")

for trial_num in range(1, 16):
    run_trial(trial_num)
```

```
 Trial #    Lower    Upper     Size          Mean        Variance     Magic
       1    3,621   77,012  101,397     40331.583   448358673.446    12.013
       2    1,030   38,670  104,978     19837.612   118274763.322    11.979
       3      910  100,746  161,436     50863.410   832864656.334    11.967
       4    2,740   36,896   44,032     19849.948    97738548.624    11.936
       5    4,947   11,408   87,748      8182.547     3464328.541    12.050
       6    3,931   79,606  114,077     41779.457   479300380.117    11.948
       7    5,298   73,859  116,363     39480.076   391681820.063    12.001
       8    4,955   55,824   35,566     30417.527   217509563.899    11.897
       9    7,415    9,901   81,025      8656.040      514782.757    12.005
      10    2,628   75,904   50,343     39192.092   445470722.786    12.053
      11    6,545   51,823   78,789     29198.462   170540400.849    12.021
      12    4,178   38,195   50,325     21126.543    96317614.595    12.014
      13    4,428   63,085  159,597     33771.320   285995086.252    12.030
      14    2,747   33,008   61,260     17844.770    76358842.198    11.992
      15    6,998   60,707  158,343     33854.692   240563603.967    11.991
```

Variance of Uniform Distributions



```
Trial #    Lower    Upper     Size         Mean       Variance     Magic
      1    2,186   97,609  100,308    50061.375  763204878.817    11.931
      2    2,456   41,355   83,467    21981.261  125285980.368    12.077
      3      832   18,461   65,817     9648.839   25938232.503    11.982
      4    4,233   42,165   31,918    23231.598  119992088.765    11.991
      5    8,879   91,012  160,019    49962.086  563505796.451    11.971
      6    1,765   87,215  140,124    44436.213  606677745.464    12.036
      7    1,549   43,086   23,841    22178.161  143154389.004    12.052
      8    8,587  105,157  130,589    56981.826  777105956.238    12.001
      9    7,127   89,418   37,812    47946.706  568515233.060    11.911
     10    1,265   11,018  102,292     6142.628    7955048.841    11.957
     11    6,830   74,990  132,704    40882.409  386369369.576    12.024
     12    9,786   27,604  148,185    18702.335   26342315.791    12.052
     13      963   10,211   14,035     5572.470    7251379.077    11.794
     14    5,717    9,443   23,348     7581.759    1146793.735    12.106
     15    2,533   29,988  135,261    16234.108   62987583.045    11.967
```

- Every set had a different lower and upper limit, size, mean, and variance… yet the magic number was ~12 for all of them!
- Why would Mother Nature choose 12 for this magic number? What is so special about 12? Why not pick a nice even 10?
- Boundless natural curiosity is what makes a good scientist…

Variance of the Uniform Distribution

$$\boxed{\sigma^2 = \frac{1}{n}\sum_{i=i}^{n}(x_i - \mu)^2 \;\; where \;\; \mu = \frac{1}{n}\sum_{i=1}^{n} x_i}$$

The *expected* value $(\mathbb{E})$ of a random variable $\boldsymbol{X}$ is its <u>mean</u> value (μ)

$$\mathbb{E}(X) = \mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Variance (σ^2) is the <u>mean</u> difference *squared* between every $\boldsymbol{X}$ and its $\mathbb{E}(X)$

$$\sigma^2 = \mathbb{E}\left[\left(X - \mathbb{E}(X)\right)^2\right]$$

The *expected* value $(\mathbb{E})$ returns a **constant** value → $\mathbb{E}(X) = \mu$

The *expected* value $(\mathbb{E})$ of a **constant** value returns that same value → $\mathbb{E}(\mu) = \mu$

$$\mathbb{E}\big(\mathbb{E}(X)\big) = \mathbb{E}(X)$$

$$\mathbb{E}\Big(\mathbb{E}\big(\mathbb{E}(X)\big)\Big) = \mathbb{E}(X)$$

$\mathbb{E}(X)$ is **idempotent**

Variance of the Uniform Distribution

$$\boxed{\sigma^2 = \frac{1}{n}\sum_{i=i}^{n}(x_i - \mu)^2 \ \ where \ \ \mu = \frac{1}{n}\sum_{i=1}^{n} x_i}$$

$$\mu = \mathbb{E}(X) = \frac{1}{n}\sum_{i=1}^{n} x_i$$

$$\sigma^2 = \mathbb{E}\left[\left(X - \mathbb{E}(X)\right)^2\right]$$

$$\mathbb{E}(\mu) = \mu$$

$$\mathbb{E}\big(\mathbb{E}(X)\big) = \mathbb{E}(X)$$

$$\sigma^2 = \mathbb{E}\left[\left(X - \mathbb{E}(X)\right)^2\right]$$ FOIL

$$\sigma^2 = \boxed{\mathbb{E}}[\boxed{X^2} - \boxed{2X\mathbb{E}(X)} + \boxed{\mathbb{E}(X)^2}]$$

Note: $\mathbb{E}(x)$ is a distributive linear operator

$$\sigma^2 = \mathbb{E}(X^2) - \mathbb{E}\big(2X\mathbb{E}(X)\big) + \mathbb{E}(\mathbb{E}(X)^2)$$

$$\sigma^2 = \mathbb{E}(X^2) - 2\mathbb{E}(X)\mathbb{E}(X) + \mathbb{E}(X)^2$$

$$\sigma^2 = \mathbb{E}(X^2) - 2\mathbb{E}(X)^2 + \mathbb{E}(X)^2$$

$$\sigma^2 = \mathbb{E}(X^2) - \mathbb{E}(X)^2$$

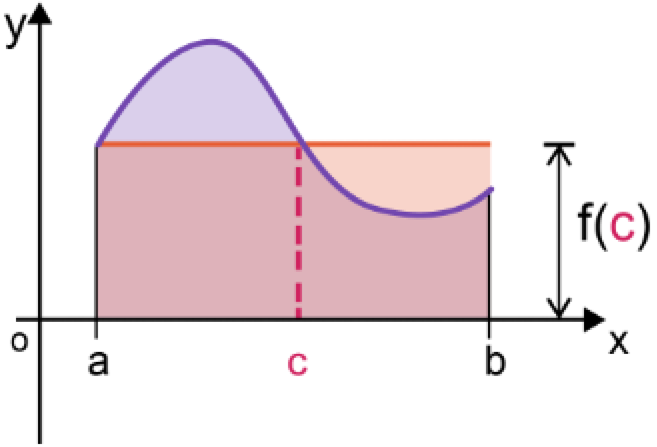
$$\boxed{\sigma^2 = \mathbb{E}(X^2) - \mu^2}$$

$$\sigma^2 = \left(\frac{1}{n}\sum_{i=1}^{n} x_i^2\right) - \mu^2$$

Faster because only one subtraction is required!

Variance of the Uniform Distribution

$f(c)$ = the average value of the function



Mean Value Theorem (*Integrals*)

$$Area_{red} = Area_{curve}$$

$$Area_{red} = f(c) \times (b-a)$$

$$Area_{curve} = \int_a^b f(x)\,dx$$

$$f(c) \times (b-a) = \int_a^b f(x)\,dx$$

$$f(c) = \frac{1}{(b-a)} \int_a^b f(x)\,dx$$

$$f(c) = \mu = \mathbb{E}(X)$$

Random Variable (Uniform Distribution)

Discrete: $\mathbb{E}(X) = \frac{1}{n}\sum_{i=1}^{n} x_i$

Continuous: $\mathbb{E}(X) = \frac{1}{(b-a)} \int_a^b x\,dx$

Variance of the Uniform Distribution

Moment Generating Functions

$$\mathbb{E}(\boxed{X}) = \frac{1}{(b-a)}\int_a^b \boxed{x}\,dx$$

$$\mathbb{E}(\boxed{X^2}) = \frac{1}{(b-a)}\int_a^b \boxed{x^2}\,dx$$

$$\sigma^2 = \mathbb{E}(X^2) - \mu^2$$

$$\mu = \frac{1}{b-a}\int_a^b x\,dx = \frac{1}{b-a}\left(\left.\frac{x^2}{2}\right|_a^b\right) = \frac{b+a}{2}$$

$$\mathbb{E}(X^2) = \frac{1}{b-a}\int_a^b x^2\,dx = \frac{1}{b-a}\left(\left.\frac{x^3}{3}\right|_a^b\right) = \frac{b^2+ab+a^2}{3}$$

Variance of the Uniform Distribution

Moment Generating Functions

$$\boxed{12 = \frac{(\textit{upper\_limit} \; - \; \textit{lower\_limit})^{2}}{\textit{variance}}}$$

$$\mathbb{E}(X) = \frac{1}{(b-a)} \int_a^b x \, dx$$

$$\mathbb{E}(X^2) = \frac{1}{(b-a)} \int_a^b x^2 \, dx$$

$$\boxed{\sigma^2 = \mathbb{E}(X^2) - \mu^2}$$

$$\mu = \frac{1}{b-a} \int_a^b x \, dx = \frac{1}{b-a} \left(\frac{x^2}{2} \bigg|_a^b \right) = \frac{b+a}{2}$$

$$\mathbb{E}(X^2) = \frac{1}{b-a} \int_a^b x^2 \, dx = \frac{1}{b-a} \left(\frac{x^3}{3} \bigg|_a^b \right) = \frac{b^2+ab+a^2}{3}$$

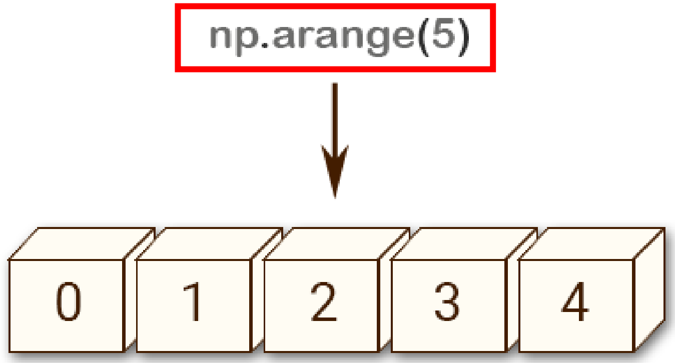
This is the **second central moment** of a *continuous* uniform distribution

$$\boxed{\sigma^2 = \mathbb{E}(X^2) - \mu^2} = \frac{b^2+ab+a^2}{3} - \left(\frac{b+a}{2} \right)^2 = \boxed{\frac{(b-a)^2}{12}}$$

Variance

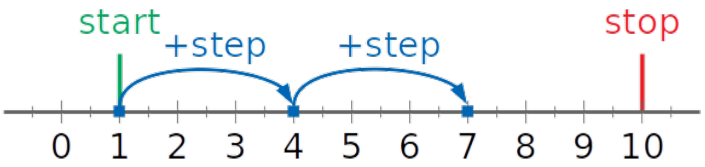
Create a **Numpy** **Array** from a **Range**

Creates a "street" of *mailboxes* where the **value** inside each mailbox follows the requested **range**

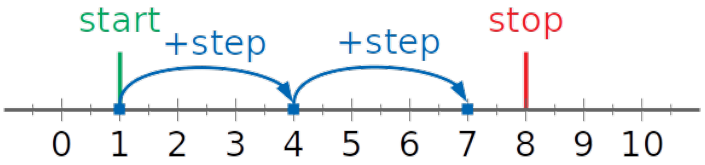


The *default values* are **start** = **0** and **step = 1**
The stop value is exclusive

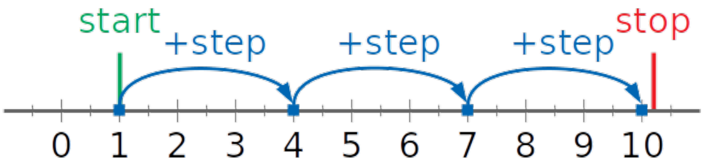
```
>>> np.arange(1, 10, 3)
array([1, 4, 7])
```



```
>>> np.arange(1, 8, 3)
array([1, 4, 7])
```



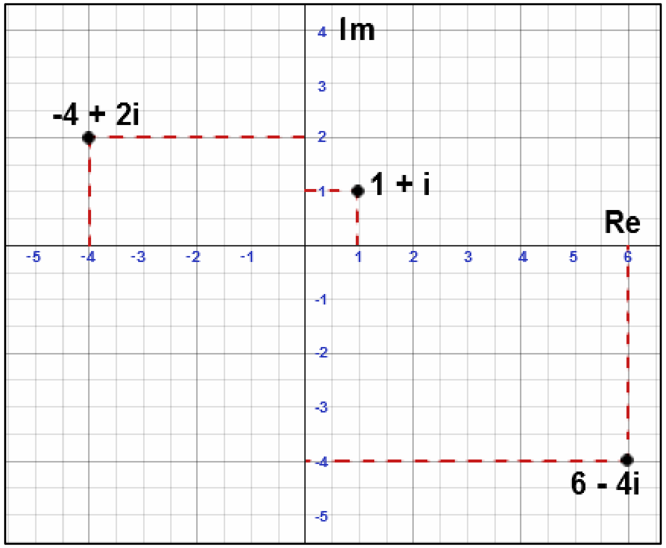
```
>>> np.arange(1, 10.1, 3)
array([1., 4., 7., 10.])
```



Complex Numbers

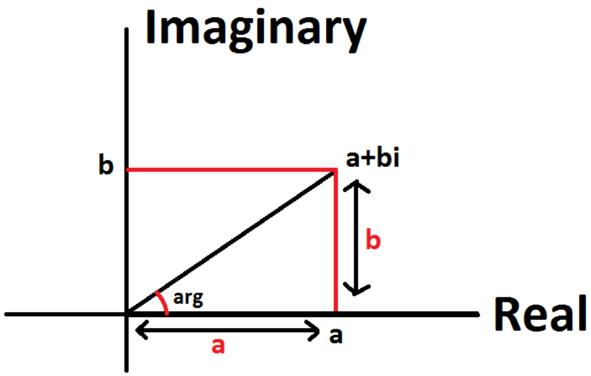
$i = \sqrt{-1}$

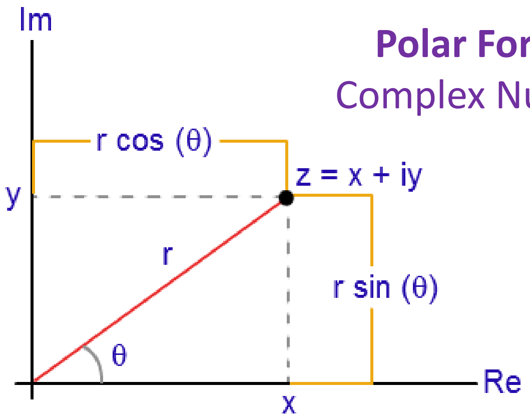
$i^2 = -1$



Argand Diagram

Jean-Robert Argand
(1768-1822)





Polar Form of
Complex Numbers

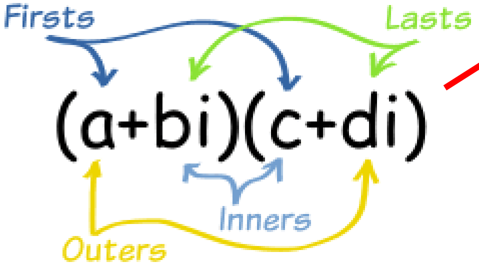
Complex Algebra

Sum: $(4+3i)+(5-4i)=(4+5)+(3-4)i$
$$=9-i$$

Difference: $(4+3i)-(5-4i)=(4-5)+\big(3-(-4)\big)i$
$$=-1+7i$$

Product: $(4+3i)(5-4i)=20-16i+15i-12i^2$
$$=20-i+12$$
$$=32-i$$

$i^2 = -1$



Complex Algebra

Division: $\frac{(4+3i)}{(5-4i)}$

$$\frac{(4+3i)}{(5-4i)} = \frac{(4+3i)}{(5-4i)} \times \frac{(5+4i)}{(5+4i)} = \frac{(8+31i)}{41}$$

Complex Conjugate

$$= \frac{8}{41} + \frac{31}{41}i$$

Euler’s Identity

- Calculate an approximation of $\boldsymbol{e^{z}}$ where $\mathbf{z} \in \mathbb{C}$, using its Taylor Series expansion to **20 terms**

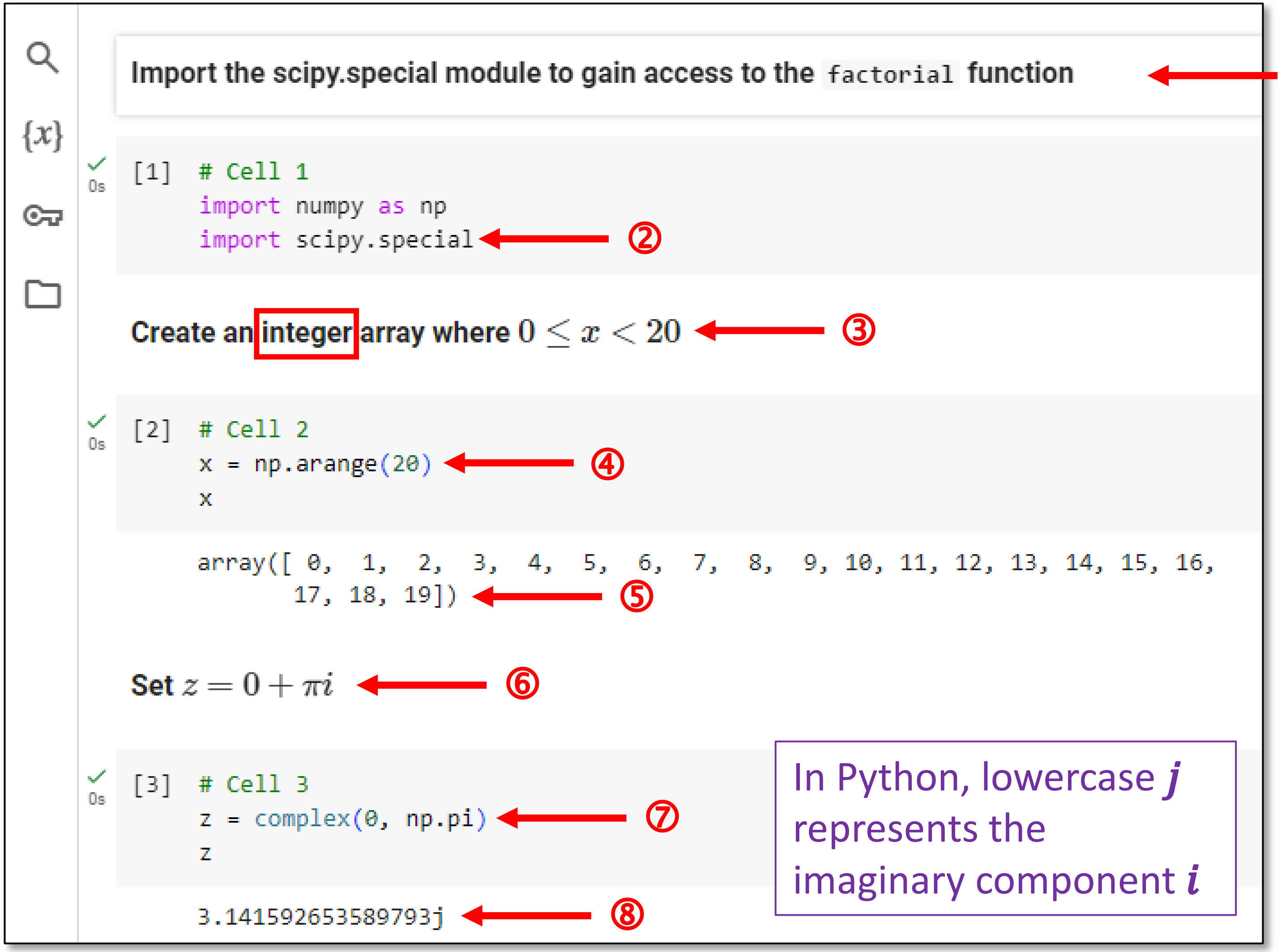
$$\boldsymbol{e^{z}} = 1 + z + \frac{z^2}{2!} + \frac{z^3}{3!} + \frac{z^4}{4!} + \frac{z^5}{5!} + \frac{z^6}{6!} + \frac{z^7}{7!} + \cdots$$

- Use the above ***power series*** to display the value of $\boldsymbol{e^{\pi i}}$

$$(e^{z}\ where\ z = 0 + \boldsymbol{\pi i})$$

- Notice the *denominators* grow at a **factorial** rate
- Fortunately, in Python the size of an integer is not restricted to a fixed number of number of bits
- In Python an **int** can expand in size to the limit of the available memory!

Run euler_identity.ipynb – **Cells 1…3**



Import the scipy.special module to gain access to the `factorial` function ①

```
[1] # Cell 1
    import numpy as np
    import scipy.special
```

②

Create an integer array where $0 \leq x < 20$ ③

```
[2] # Cell 2
    x = np.arange(20)
    x
```

④

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19])
```

⑤

Set $z = 0 + \pi i$ ⑥

```
[3] # Cell 3
    z = complex(0, np.pi)
    z
```

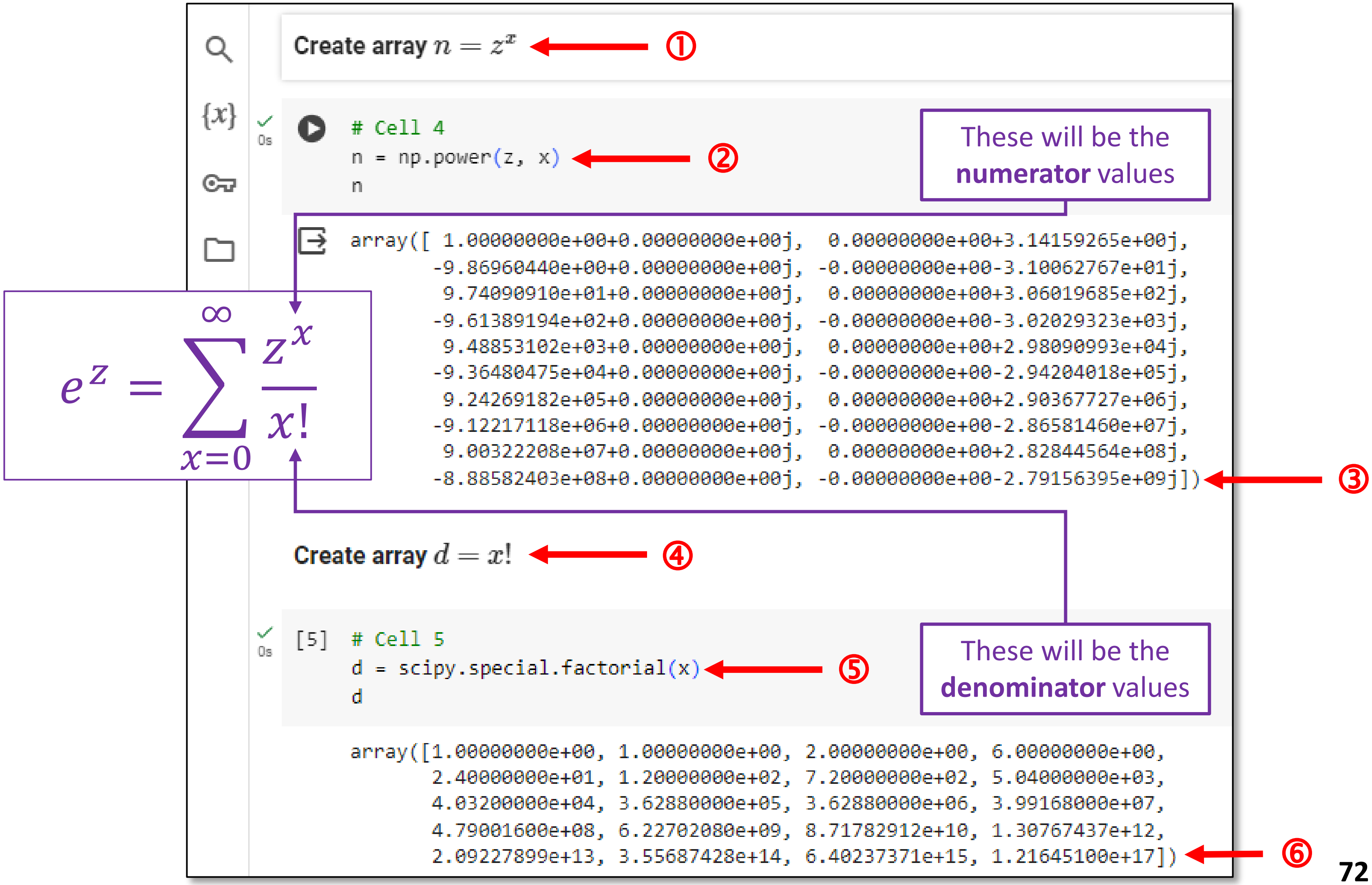
⑦

```
3.141592653589793j
```

⑧

In Python, lowercase ***j*** represents the imaginary component ***i***

Run euler_identity.ipynb – **Cells 4…5**



Create array $n = z^x$ ①

```
# Cell 4
n = np.power(z, x)
n
```
②

These will be the **numerator** values

```
array([ 1.00000000e+00+0.00000000e+00j,  0.00000000e+00+3.14159265e+00j,
       -9.86960440e+00+0.00000000e+00j, -0.00000000e+00-3.10062767e+01j,
        9.74090910e+01+0.00000000e+00j,  0.00000000e+00+3.06019685e+02j,
       -9.61389194e+02+0.00000000e+00j, -0.00000000e+00-3.02029323e+03j,
        9.48853102e+03+0.00000000e+00j,  0.00000000e+00+2.98090993e+04j,
       -9.36480475e+04+0.00000000e+00j, -0.00000000e+00-2.94204018e+05j,
        9.24269182e+05+0.00000000e+00j,  0.00000000e+00+2.90367727e+06j,
       -9.12217118e+06+0.00000000e+00j, -0.00000000e+00-2.86581460e+07j,
        9.00322208e+07+0.00000000e+00j,  0.00000000e+00+2.82844564e+08j,
       -8.88582403e+08+0.00000000e+00j, -0.00000000e+00-2.79156395e+09j])
```
③

$$e^z = \sum_{x=0}^{\infty} \frac{z^x}{x!}$$

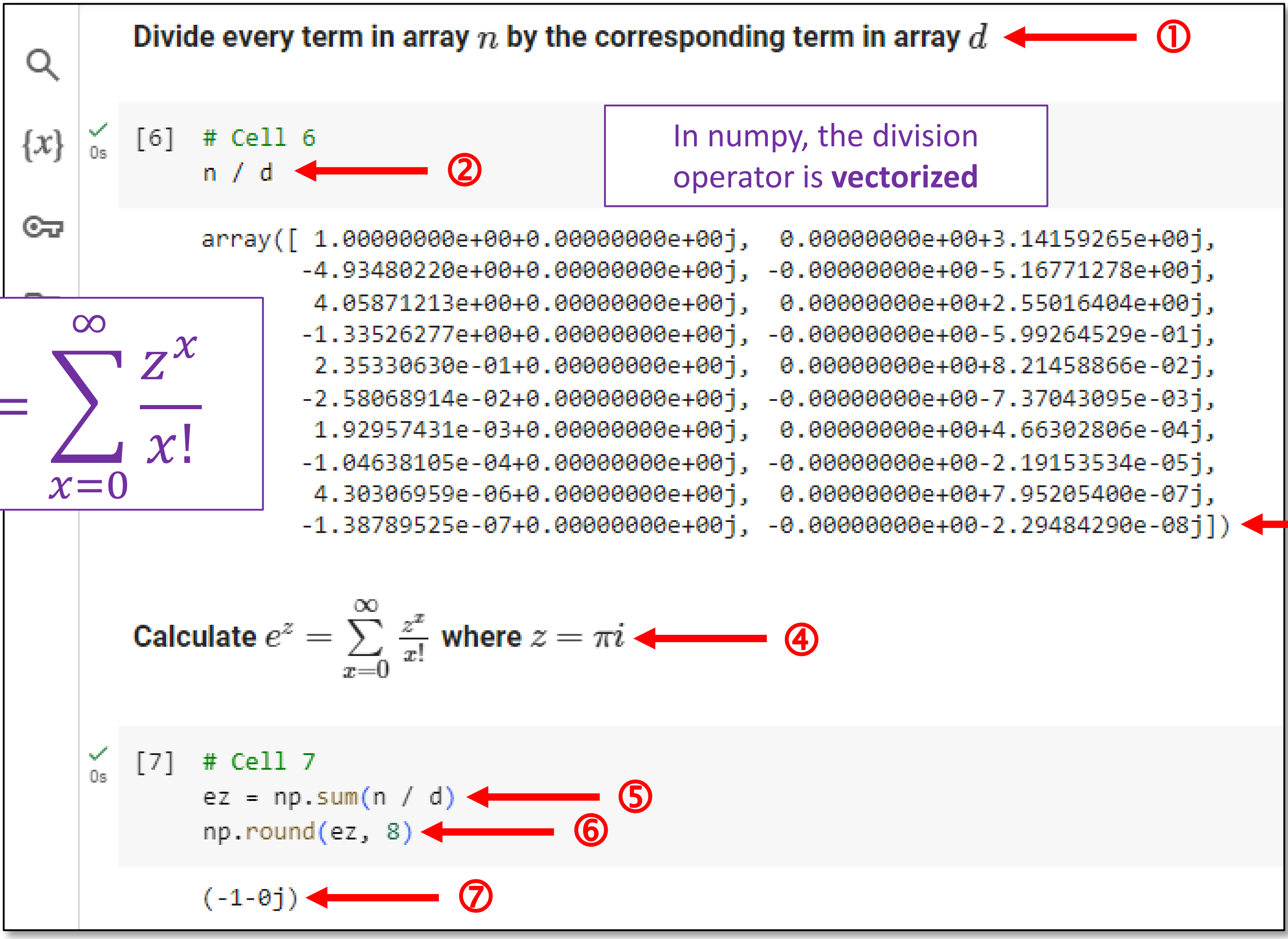
Create array $d = x!$ ④

```
[5] # Cell 5
    d = scipy.special.factorial(x)
    d
```
⑤

These will be the **denominator** values

```
array([1.00000000e+00, 1.00000000e+00, 2.00000000e+00, 6.00000000e+00,
       2.40000000e+01, 1.20000000e+02, 7.20000000e+02, 5.04000000e+03,
       4.03200000e+04, 3.62880000e+05, 3.62880000e+06, 3.99168000e+07,
       4.79001600e+08, 6.22702080e+09, 8.71782912e+10, 1.30767437e+12,
       2.09227899e+13, 3.55687428e+14, 6.40237371e+15, 1.21645100e+17])
```
⑥

Run euler_identity.ipynb – Cells 6…7



Divide every term in array n by the corresponding term in array d

```
[6] # Cell 6
    n / d
```

```
array([ 1.00000000e+00+0.00000000e+00j,  0.00000000e+00+3.14159265e+00j,
       -4.93480220e+00+0.00000000e+00j, -0.00000000e+00-5.16771278e+00j,
        4.05871213e+00+0.00000000e+00j,  0.00000000e+00+2.55016404e+00j,
       -1.33526277e+00+0.00000000e+00j, -0.00000000e+00-5.99264529e-01j,
        2.35330630e-01+0.00000000e+00j,  0.00000000e+00+8.21458866e-02j,
       -2.58068914e-02+0.00000000e+00j, -0.00000000e+00-7.37043095e-03j,
        1.92957431e-03+0.00000000e+00j,  0.00000000e+00+4.66302806e-04j,
       -1.04638105e-04+0.00000000e+00j, -0.00000000e+00-2.19153534e-05j,
        4.30306959e-06+0.00000000e+00j,  0.00000000e+00+7.95205400e-07j,
       -1.38789525e-07+0.00000000e+00j, -0.00000000e+00-2.29484290e-08j])
```

Calculate $e^z = \sum_{x=0}^{\infty} \frac{z^x}{x!}$ where $z = \pi i$

```
[7] # Cell 7
    ez = np.sum(n / d)
    np.round(ez, 8)
```

```
(-1-0j)
```

In numpy, the division operator is **vectorized**

$$e^z = \sum_{x=0}^{\infty} \frac{z^x}{x!}$$

Euler’s Identity

$$e^{\pi i} = \sum_{x=0}^{\infty} \frac{(\pi i)^x}{x!} = 1 + \pi i + \frac{(\pi i)^2}{2!} + \frac{(\pi i)^3}{3!} + \frac{(\pi i)^4}{4!} + \frac{(\pi i)^5}{5!} + \frac{(\pi i)^6}{6!} + \frac{(\pi i)^7}{7!} + \cdots$$

$$e^{\pi i} = \sum_{x=0}^{\infty} \frac{(\pi i)^x}{x!} = 1 + \pi i - \frac{\pi^2}{2} - \frac{\pi^3 i}{6} + \frac{\pi^4}{24} + \frac{\pi^5 i}{120} - \frac{\pi^6}{720} - \frac{\pi^7 i}{5040} + \cdots$$

$$e^{\pi i} = \sum_{x=0}^{\infty} \frac{(\pi i)^x}{x!} = \overset{\cos(\pi)}{\boxed{1 - \frac{\pi^2}{2} + \frac{\pi^4}{24} - \frac{\pi^6}{720}}} + \overset{\sin(\pi)}{\boxed{\left(\pi - \frac{\pi^3}{6} + \frac{\pi^5}{120} - \frac{\pi^7}{5040}\right)}} i + \cdots$$

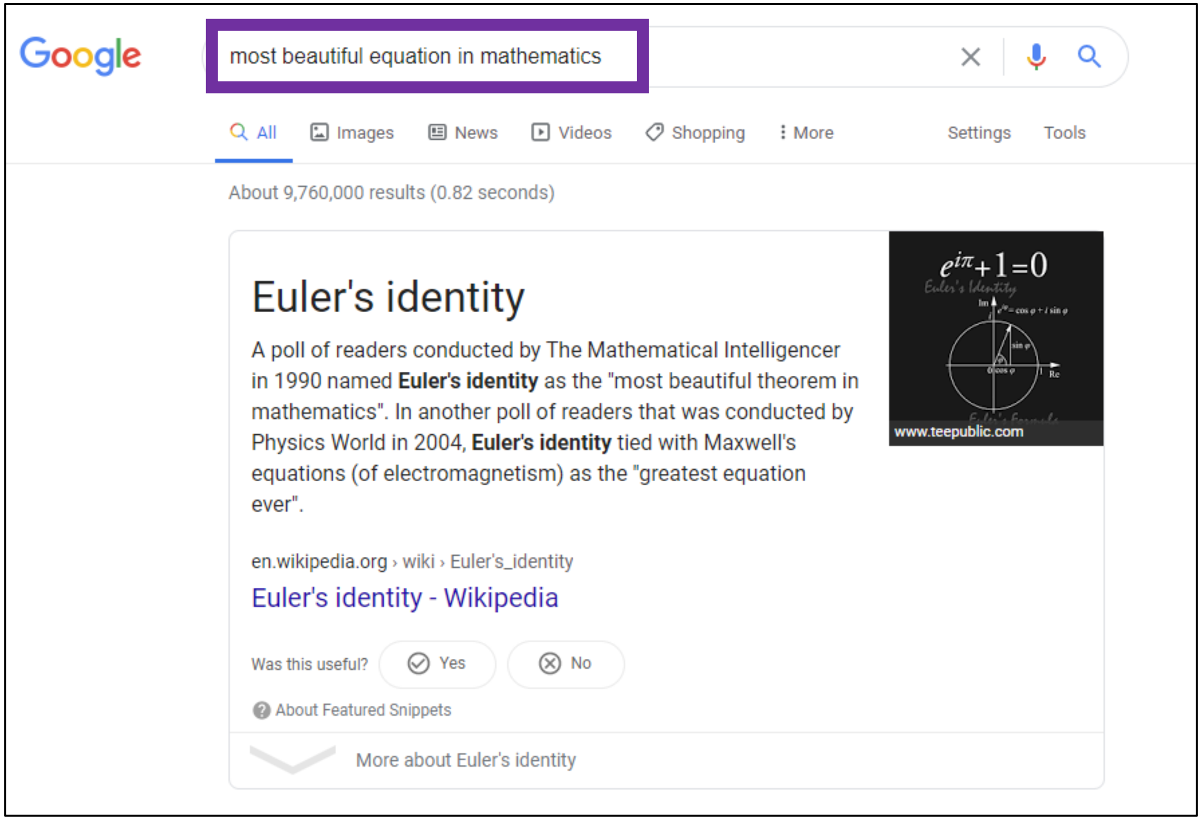
$$e^{\pi i} = -1$$

$$e^{\pi i} + 1 = 0$$

Euler

Euler’s Identity

$$e^{\pi i} + 1 = 0$$



most beautiful equation in mathematics

About 9,760,000 results (0.82 seconds)

Euler's identity

A poll of readers conducted by The Mathematical Intelligencer in 1990 named **Euler's identity** as the "most beautiful theorem in mathematics". In another poll of readers that was conducted by Physics World in 2004, **Euler's identity** tied with Maxwell's equations (of electromagnetism) as the "greatest equation ever".

en.wikipedia.org › wiki › Euler's_identity

Euler's identity - Wikipedia

Was this useful? Yes No

About Featured Snippets

More about Euler's identity

$$\boxed{i^i = ?}$$

$$\left(a^b\right)^c = a^{bc}$$

$$(2^3)^4 = 2^{3\times 4} = 2^{12}$$

$$e^{\pi i} = -1$$

$$-1 = e^{\pi i}$$

$$(-1)^{\frac{1}{2}} = \left(e^{\pi i}\right)^{\frac{1}{2}}$$

$$\sqrt{-1} = e^{\frac{\pi i}{2}}$$

$$i = e^{\frac{\pi i}{2}}$$

$$i^i = \left(e^{\frac{\pi i}{2}}\right)^i$$

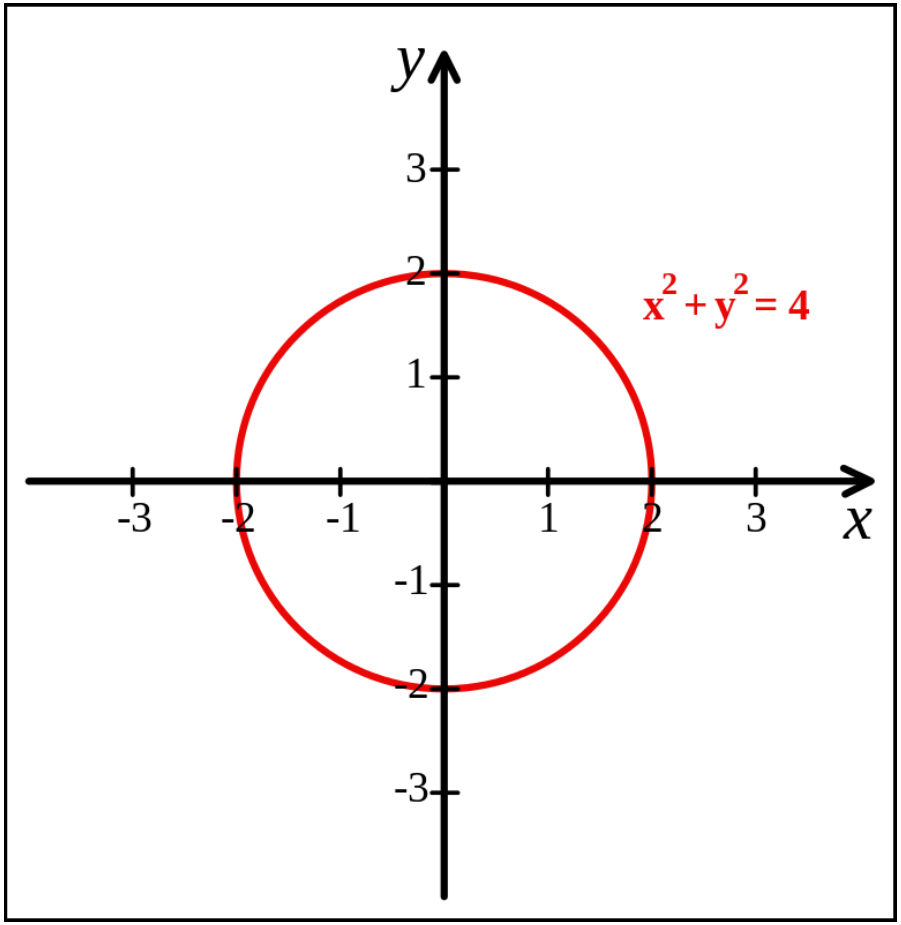
$$i^i = e^{\frac{\pi i^2}{2}}$$

$$i^i = e^{\frac{-\pi}{2}}$$

$$\boldsymbol{i^i \cong 0.20787} \in \mathbb{R}$$

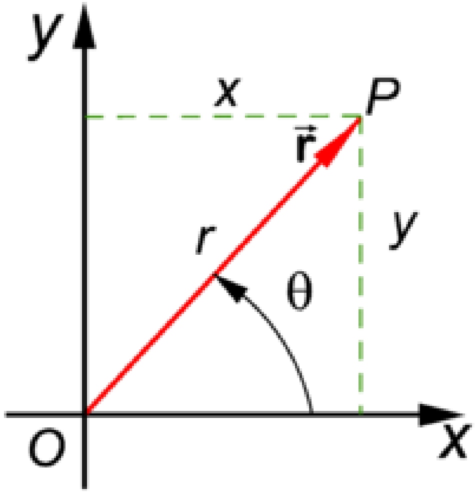
Cartesian Coordinates

Created by
René Descartes
in 1637

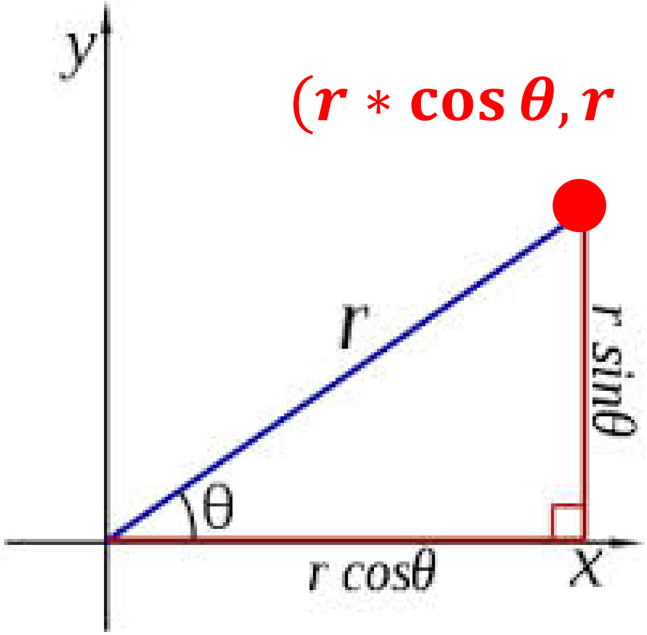


Polar Coordinates

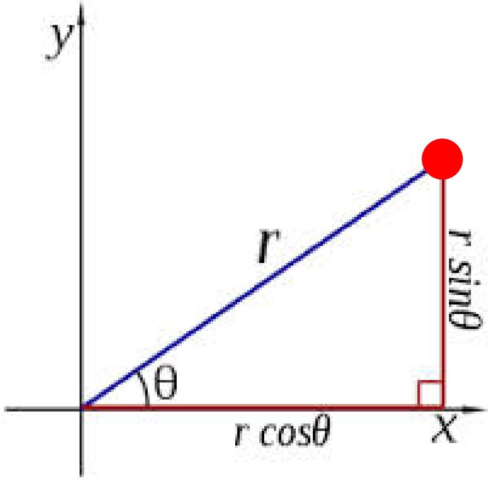
A radius and an angle (theta)
make a 2D polar coordinate

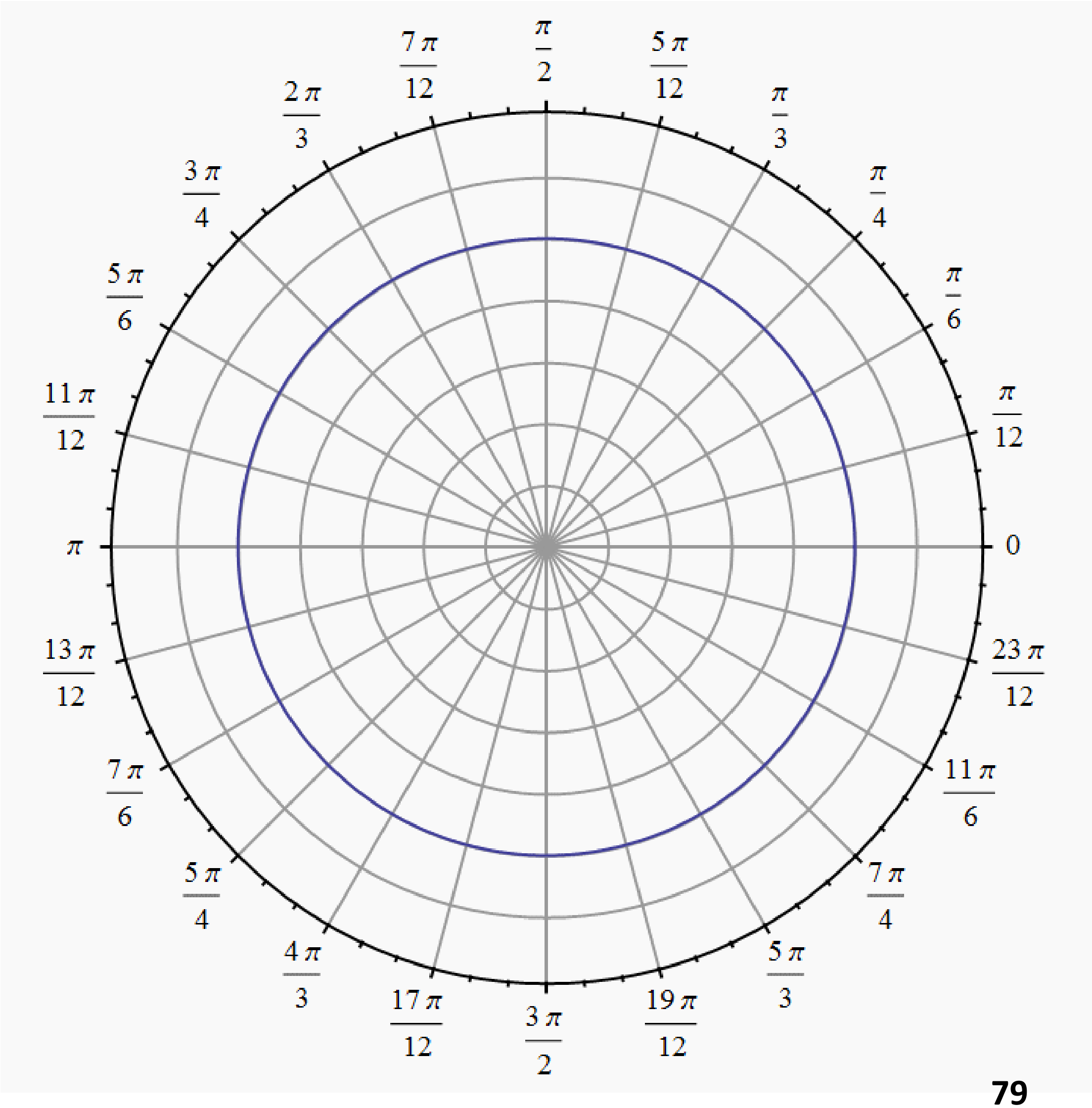


$(\boldsymbol{r} * \cos\boldsymbol{\theta}, \boldsymbol{r} * \sin\boldsymbol{\theta})$



Polar Coordinates



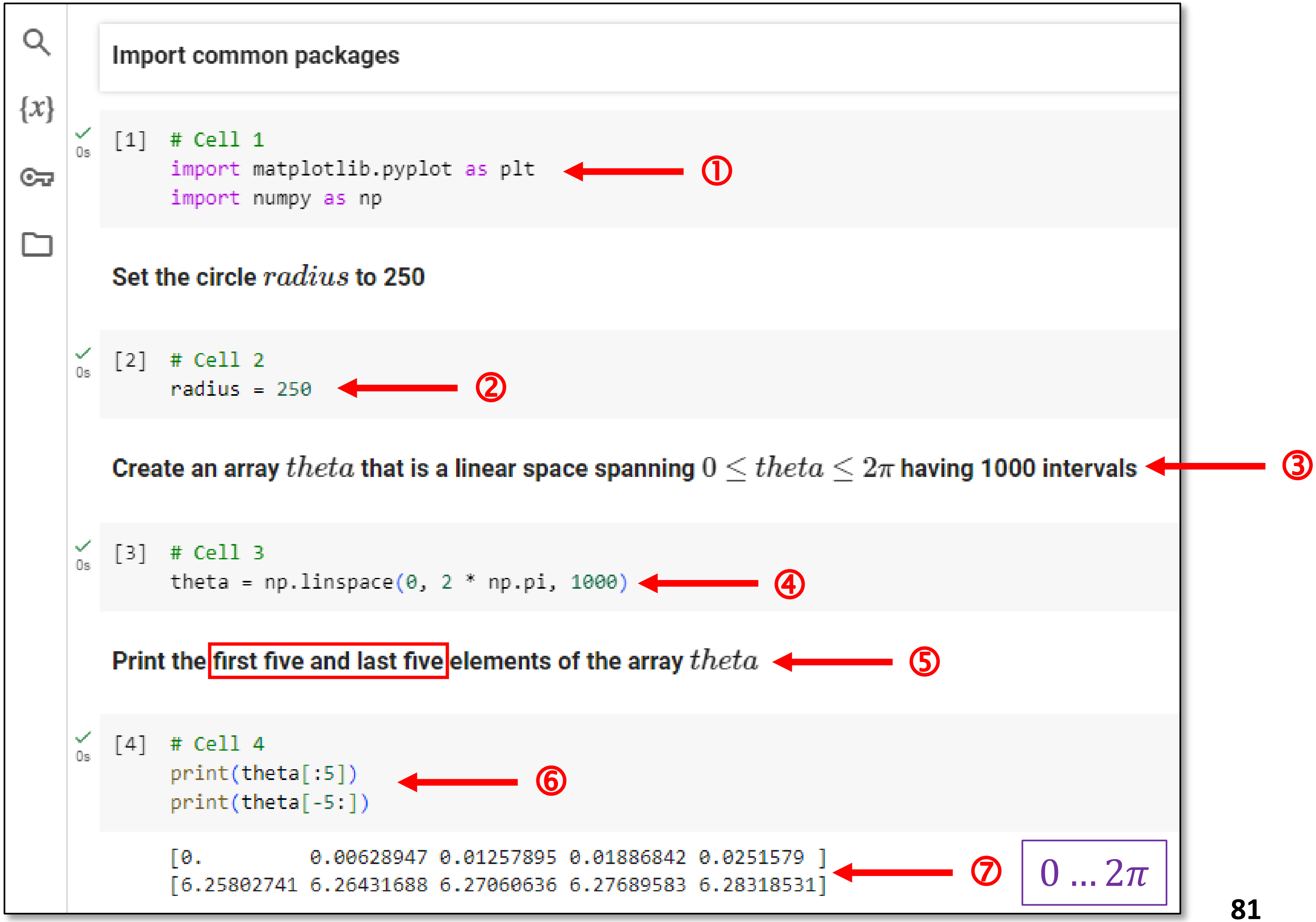


Angles are measured in **radians** ($0 \leq \theta \leq 2\pi$)

Polar to Cartesian Coordinate Conversion

- Your scientist wants you to draw a **blue** circle with a **radius of 250** *centered* at the **origin**
- Solution strategy:
  - Create a **Numpy** array of 1,000 equally spaced independent radian angle values spanning the interval $0 \leq \theta \leq 2\pi$
  - Create an array of dependent variable values - the (x, y) Cartesian coordinates - by invoking **vectorized** mathematical operators across the array of independent values
  - Have **Matplotlib** "connect the dots" between successive (x, y) Cartesian points (drawing straight line segments between them) to make the plot appear *smooth* to the unaided human eye

Run plot_circle.ipynb – Cells 1…4



Import common packages

```
[1] # Cell 1
    import matplotlib.pyplot as plt
    import numpy as np
```

Set the circle $radius$ to 250

```
[2] # Cell 2
    radius = 250
```

Create an array $theta$ that is a linear space spanning $0 \leq theta \leq 2\pi$ having 1000 intervals

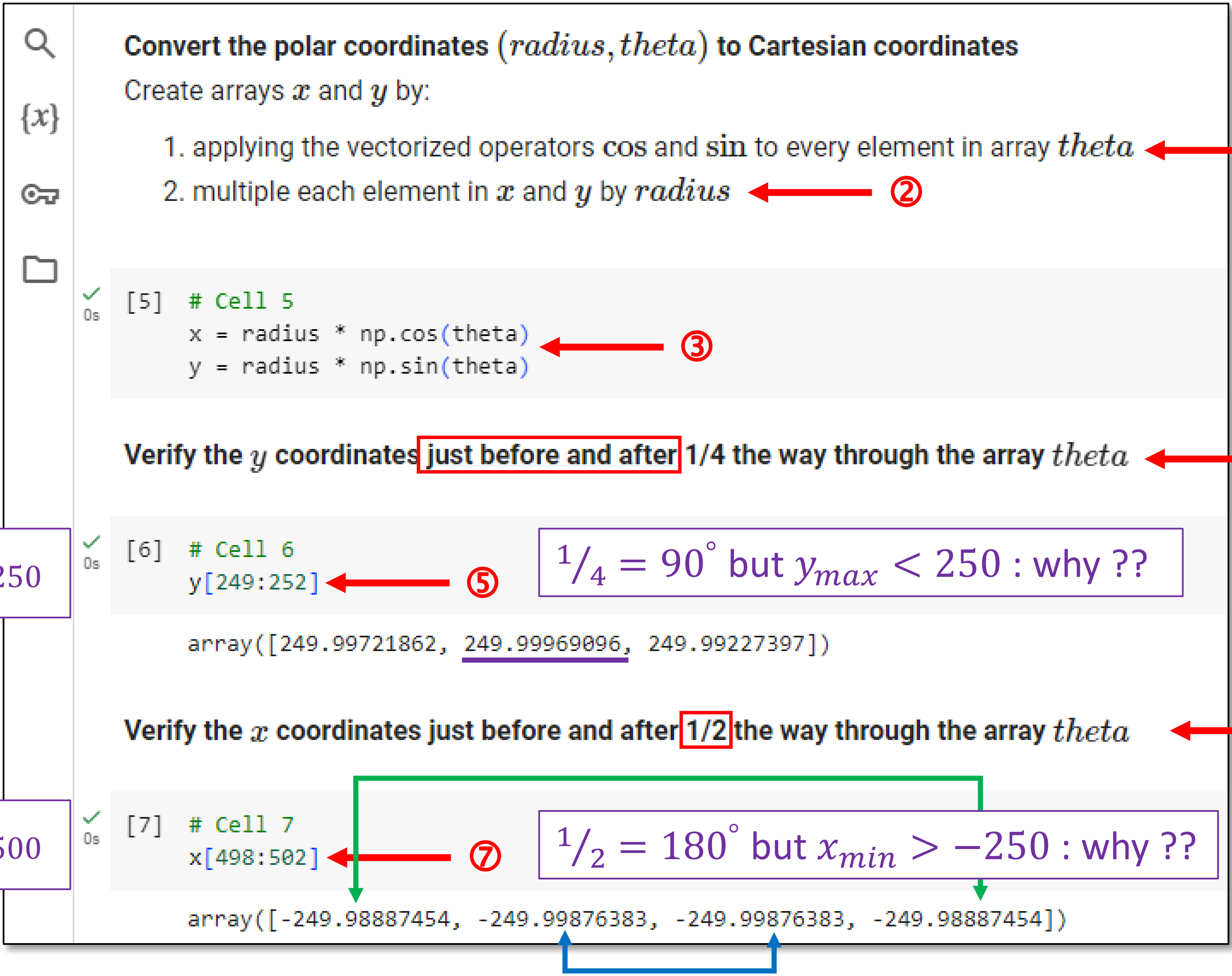
```
[3] # Cell 3
    theta = np.linspace(0, 2 * np.pi, 1000)
```

Print the first five and last five elements of the array $theta$

```
[4] # Cell 4
    print(theta[:5])
    print(theta[-5:])
```

```
    [0.         0.00628947 0.01257895 0.01886842 0.0251579 ]
    [6.25802741 6.26431688 6.27060636 6.27689583 6.28318531]
```

Run plot_circle.ipynb – **Cells 5…7**



Convert the polar coordinates $(radius, theta)$ to Cartesian coordinates

Create arrays x and y by:

1. applying the vectorized operators cos and sin to every element in array $theta$ ①
2. multiple each element in x and y by $radius$ ②

```
[5]  # Cell 5
     x = radius * np.cos(theta)
     y = radius * np.sin(theta)
```

③

Verify the y coordinates just before and after 1/4 the way through the array $theta$ ④

$\frac{1000}{4} = 250$

```
[6]  # Cell 6
     y[249:252]
```

⑤

$^1/_4 = 90^\circ$ but $y_{max} < 250$: why ??

```
array([249.99721862, 249.99969096, 249.99227397])
```

Verify the x coordinates just before and after 1/2 the way through the array $theta$ ⑥

$\frac{1000}{2} = 500$

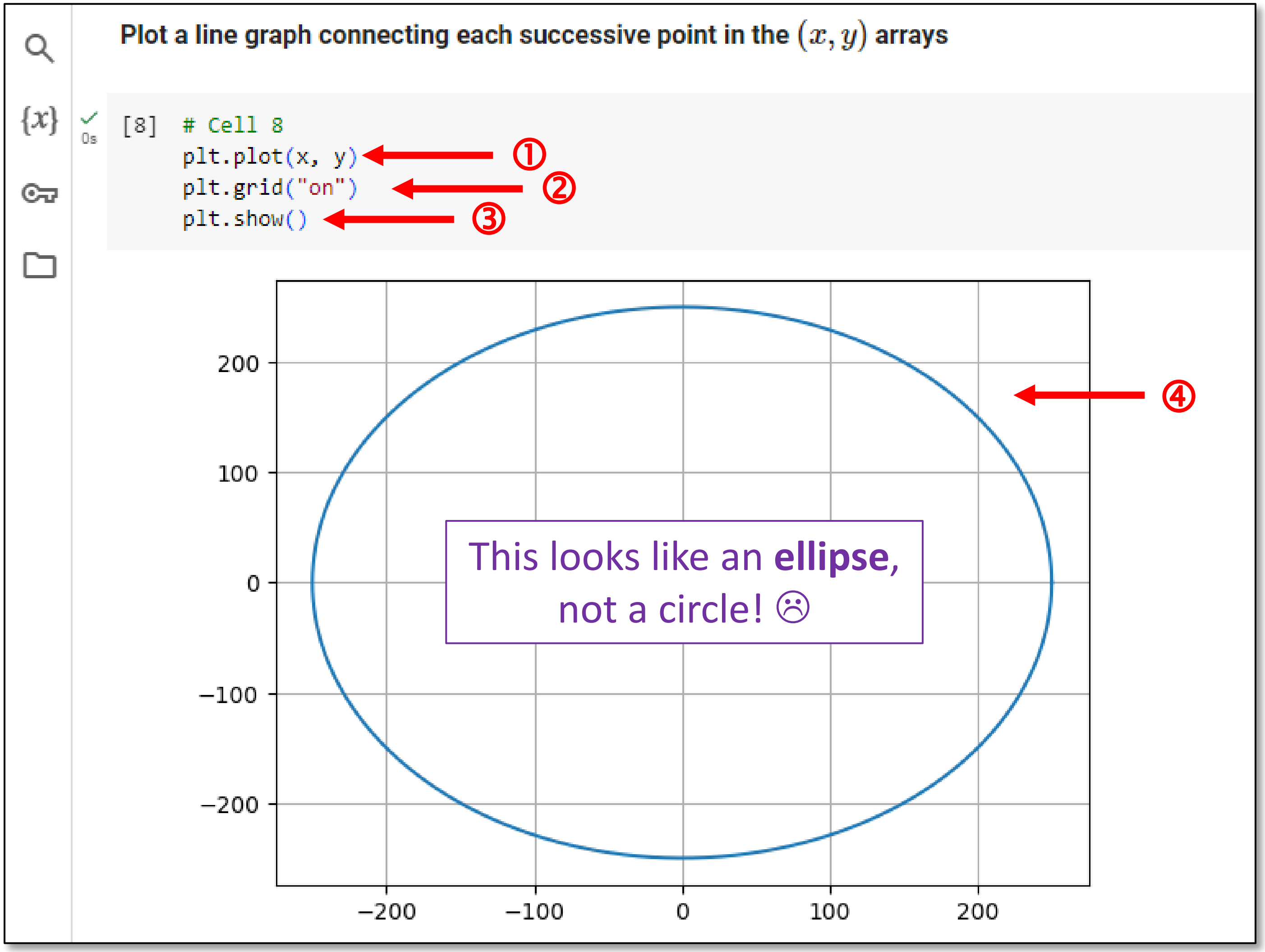
```
[7]  # Cell 7
     x[498:502]
```

⑦

$^1/_2 = 180^\circ$ but $x_{min} > -250$: why ??

```
array([-249.98887454, -249.99876383, -249.99876383, -249.98887454])
```

Run plot_circle.ipynb – **Cell 8**

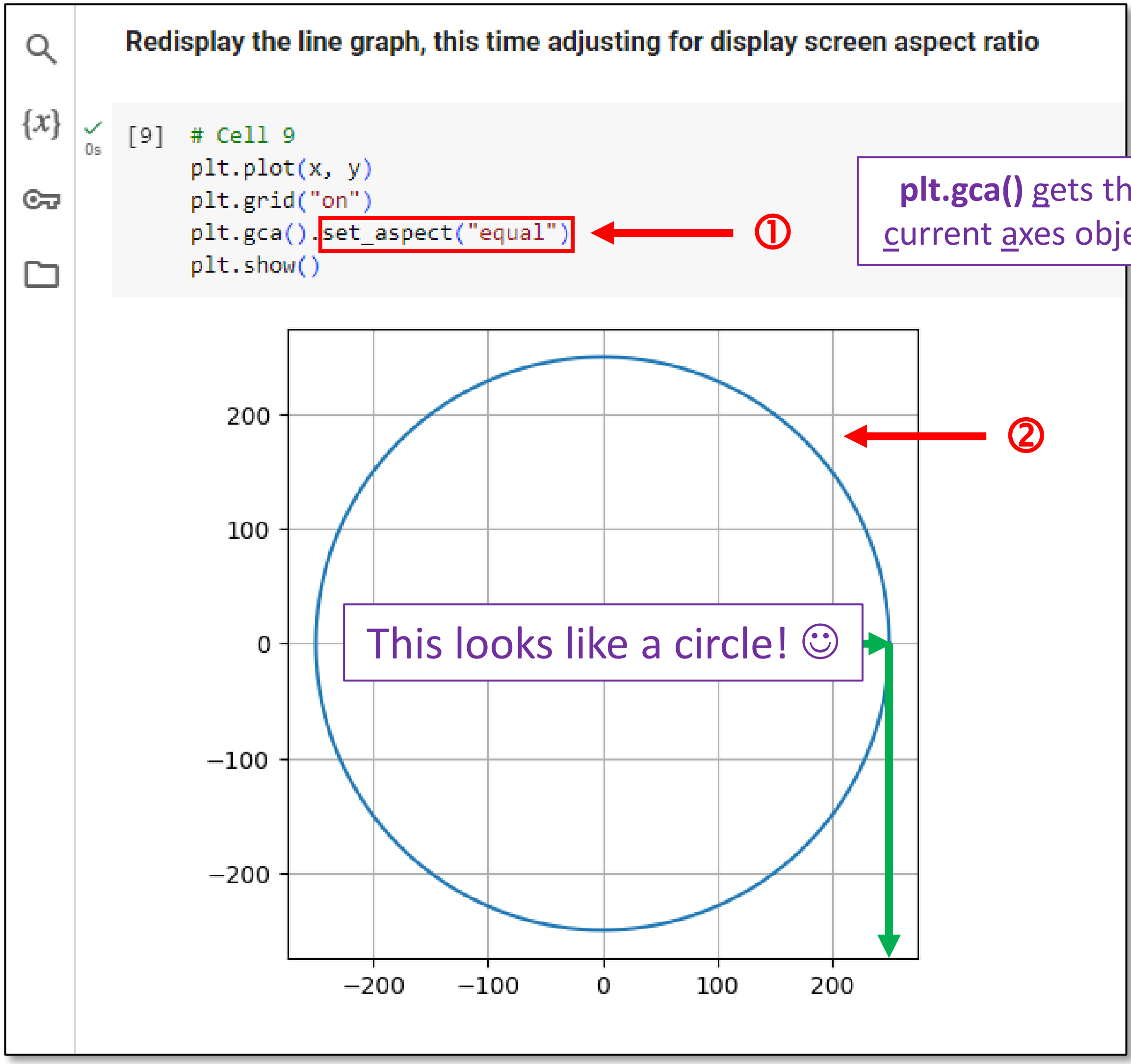


Plot a line graph connecting each successive point in the (x, y) arrays

```
[8]  # Cell 8
     plt.plot(x, y)
     plt.grid("on")
     plt.show()
```

This looks like an **ellipse**, not a circle! ☹

Run plot_circle.ipynb – **Cell 9**



Redisplay the line graph, this time adjusting for display screen aspect ratio

```
# Cell 9
plt.plot(x, y)
plt.grid("on")
plt.gca().set_aspect("equal")
plt.show()
```

①

plt.gca() gets the <u>c</u>urrent <u>a</u>xes object

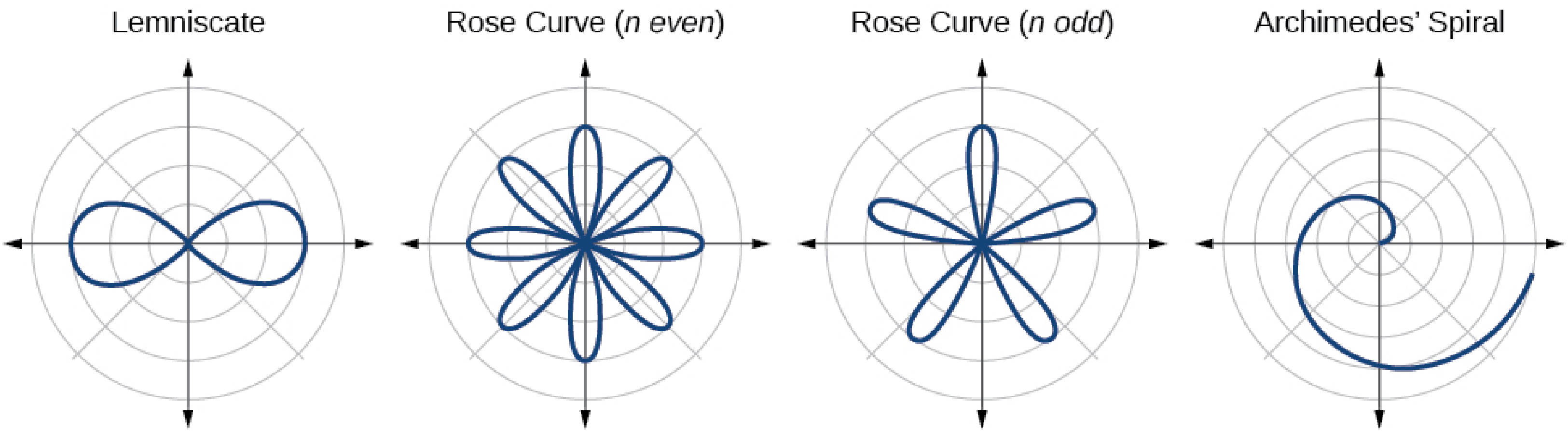
②

This looks like a circle! ☺

Parametric Curves

SPIROGRAPH
A simple and fascinating way to DRAW a million marvelous patterns
Kenner's SPIRO
For all ages... Anyone ca
Create you
No limit to
DUKE
CHALK

Parametric Curves



$r^2 = a^2\cos 2\theta$
$r^2 = a^2\sin 2\theta$
$a \neq 0$

(a)

$r = a\cos n\theta$
$r = a\sin n\theta$
n even, $2n$ petals

(b)

$r = a\cos n\theta$
$r = a\sin n\theta$
n odd, n petals

(c)

$r = \theta$
$\theta \geq 0$

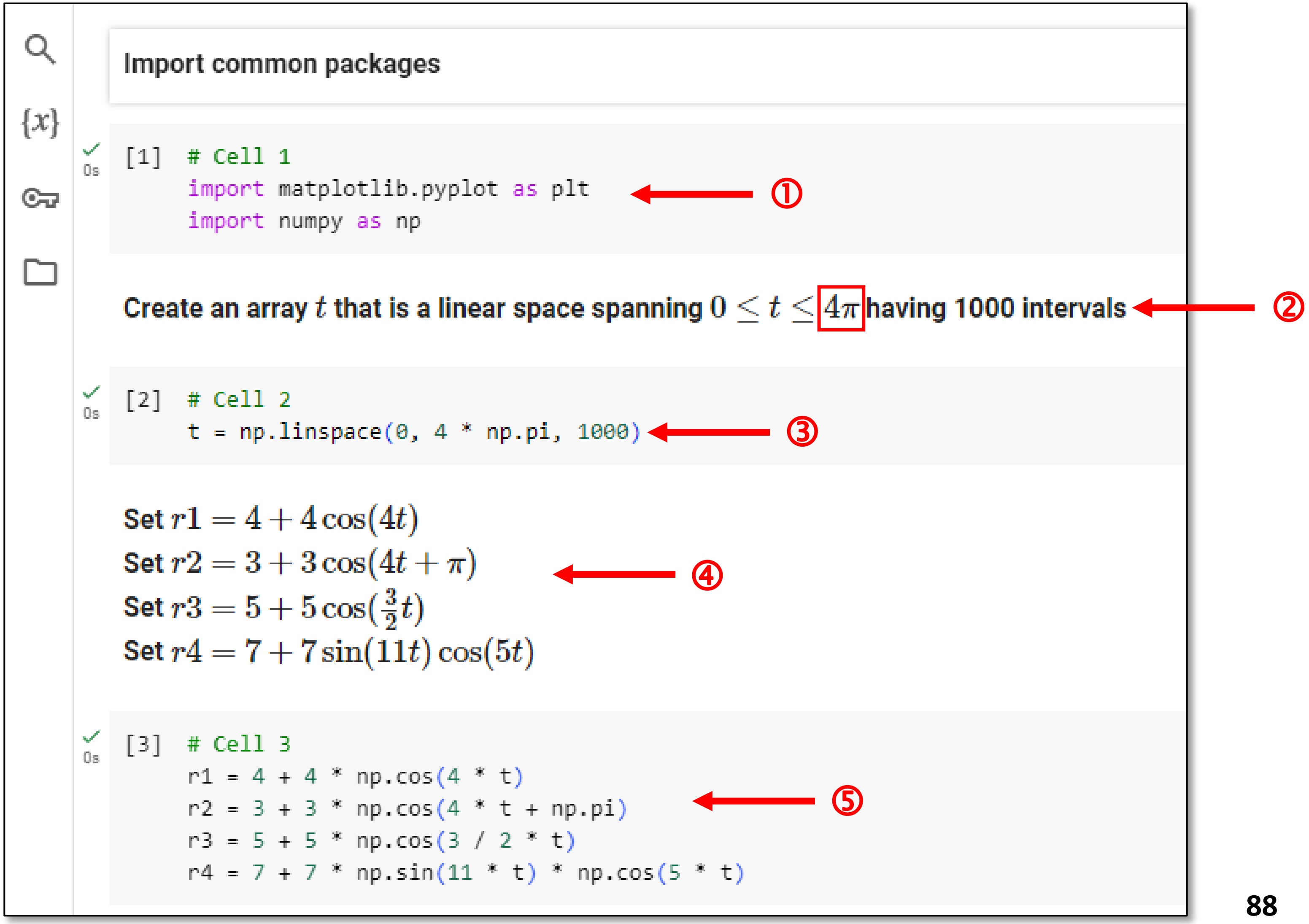
(d)

The two parameters **a, n** are used to calculate the current radius $\boldsymbol{r}$ as $\boldsymbol{\theta}$ sweeps the circle

Parametric Curves Using Polar Graphs

- Your scientist wants you to plot three **parametric curves** using the built-in **polar graph** capability of matplotlib
  - Plot $r_1 = 4 + 4\cos(4\theta)$
  - Plot $r_2 = 3 + 3\cos(4\theta + \pi)$
  - Plot $r_3 = 5 + 5\cos\left(\frac{3}{2}\theta\right)$
- Use **1,000** intervals equally spaced between $0 \leq \theta \leq 4\pi$
- Before the computer shows the plots, can you predict ahead of time what each curve will look like?
- Developing a **visual intuition** for how functions behave is a very valuable skill that will aid you in future math classes

Run plot_rose_curves.ipynb – **Cells 1…3**



Import common packages

```
[1]  # Cell 1
     import matplotlib.pyplot as plt
     import numpy as np
```

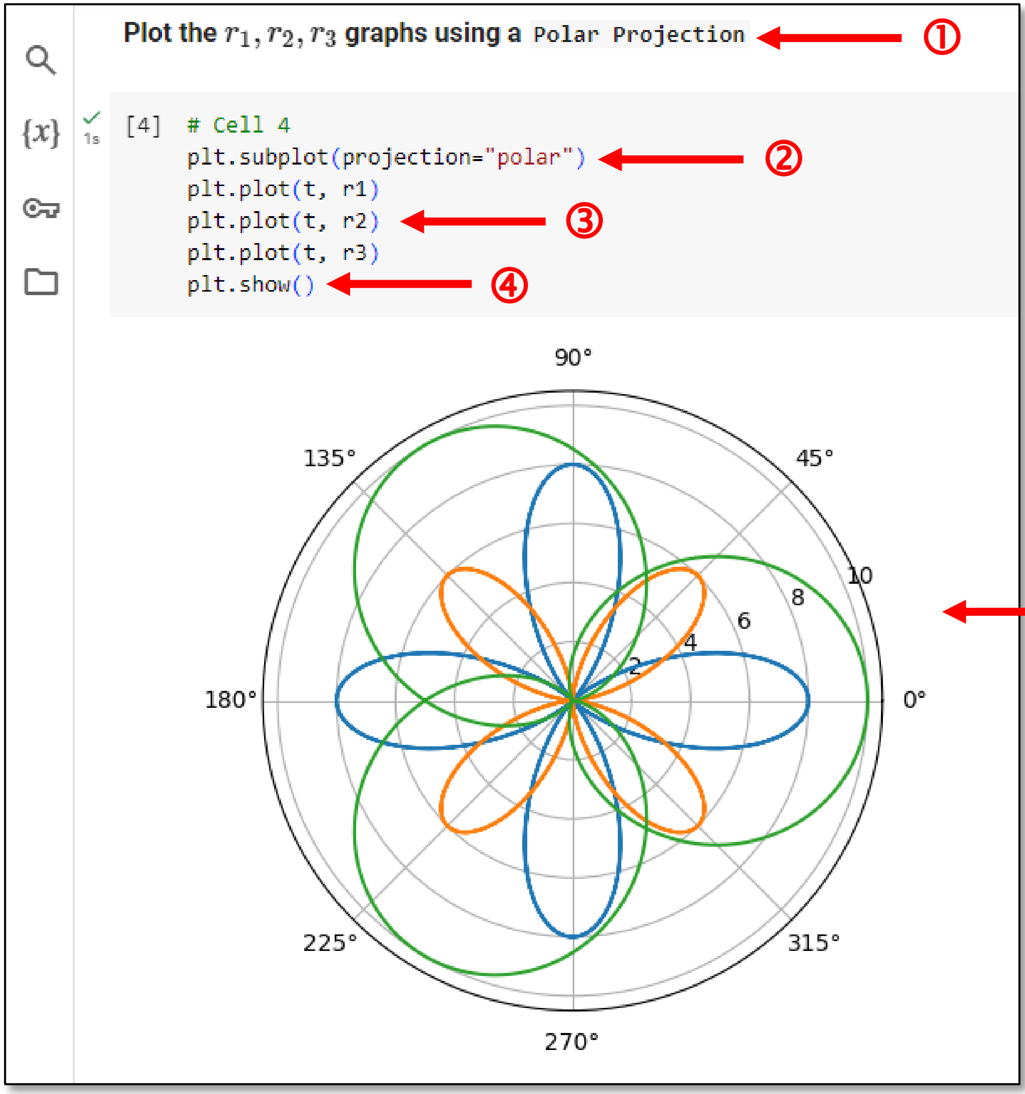
Create an array t that is a linear space spanning $0 \leq t \leq \boxed{4\pi}$ having 1000 intervals

```
[2]  # Cell 2
     t = np.linspace(0, 4 * np.pi, 1000)
```

Set $r1 = 4 + 4\cos(4t)$
Set $r2 = 3 + 3\cos(4t + \pi)$
Set $r3 = 5 + 5\cos(\frac{3}{2}t)$
Set $r4 = 7 + 7\sin(11t)\cos(5t)$

```
[3]  # Cell 3
     r1 = 4 + 4 * np.cos(4 * t)
     r2 = 3 + 3 * np.cos(4 * t + np.pi)
     r3 = 5 + 5 * np.cos(3 / 2 * t)
     r4 = 7 + 7 * np.sin(11 * t) * np.cos(5 * t)
```

Run plot_rose_curves.ipynb – **Cell 4**



Plot the r_1, r_2, r_3 graphs using a `Polar Projection` ①

```
[4]  # Cell 4
     plt.subplot(projection="polar")
     plt.plot(t, r1)
     plt.plot(t, r2)
     plt.plot(t, r3)
     plt.show()
```

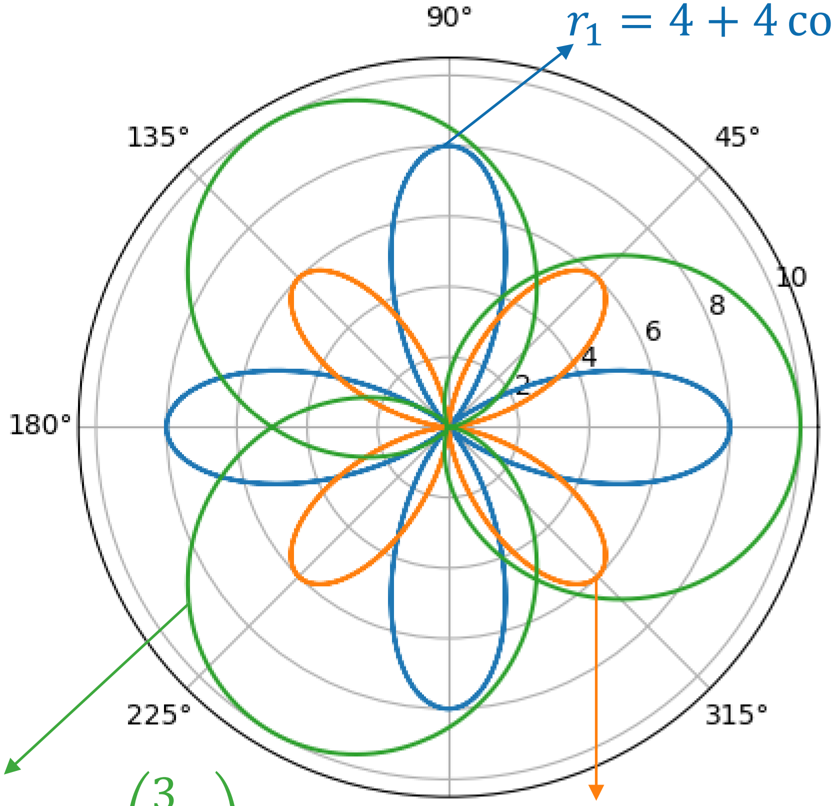
②

③

④

⑤

Parametric Curves Using Polar Graphs



$r_1 = 4 + 4\cos(4\theta)$

Why does this curve have four petals?

Why is this curve canted 120° and what did the denominator of 2 affect?

$r_3 = 5 + 5\cos\left(\frac{3}{2}\theta\right)$

Why is this curve canted 45°?

$r_2 = 3 + 3\cos(4\theta + \pi)$

The Superposition of Waves

- Even just two simple sinusoids (waves) when placed in **superposition** (*added* together) can produce very complicated results
- Your scientist wants to study the behavior of this superposition: $r_4 = 7 + 7\sin(11\theta)\cos(5\theta)$
- Plot r_4 with a **black pen** over the interval $0 \leq \theta \leq 4\pi$
- There is a **trigonometry identity** called the "angle product formula" that allows us to represent the superposition of two sinusoids **as the product of their respective wave functions**

The Superposition of Waves

$$r_4 = 7 + 7\sin(11t)\cos(5t)$$

$\theta = 11$
$\varphi = 5$

Angle Product Identity

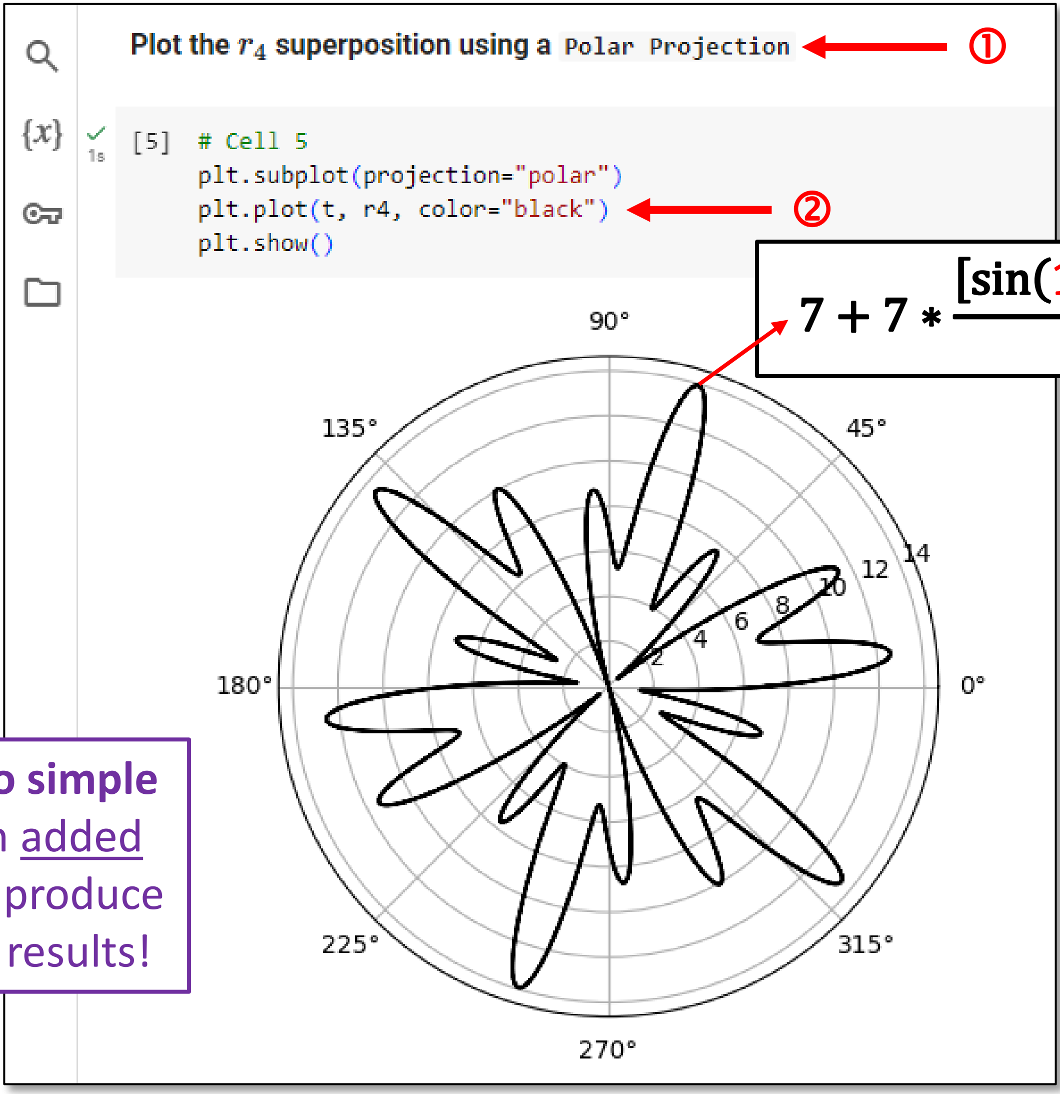
$$\sin\theta\cos\varphi = \frac{\sin(\theta+\varphi)+\sin(\theta-\varphi)}{2}$$

$$7 + 7\sin 11t \cos 5t = 7 + \frac{7}{2}[\sin(16t)\boxed{+}\sin(6t)]$$

Superposition

In classical wave theory, when waves overlap,
their **amplitudes add up linearly**

Run plot_rose_curves.ipynb – Cell 5



Plot the r_4 superposition using a `Polar Projection` ①

```
[5]  # Cell 5
     plt.subplot(projection="polar")
     plt.plot(t, r4, color="black")
     plt.show()
```

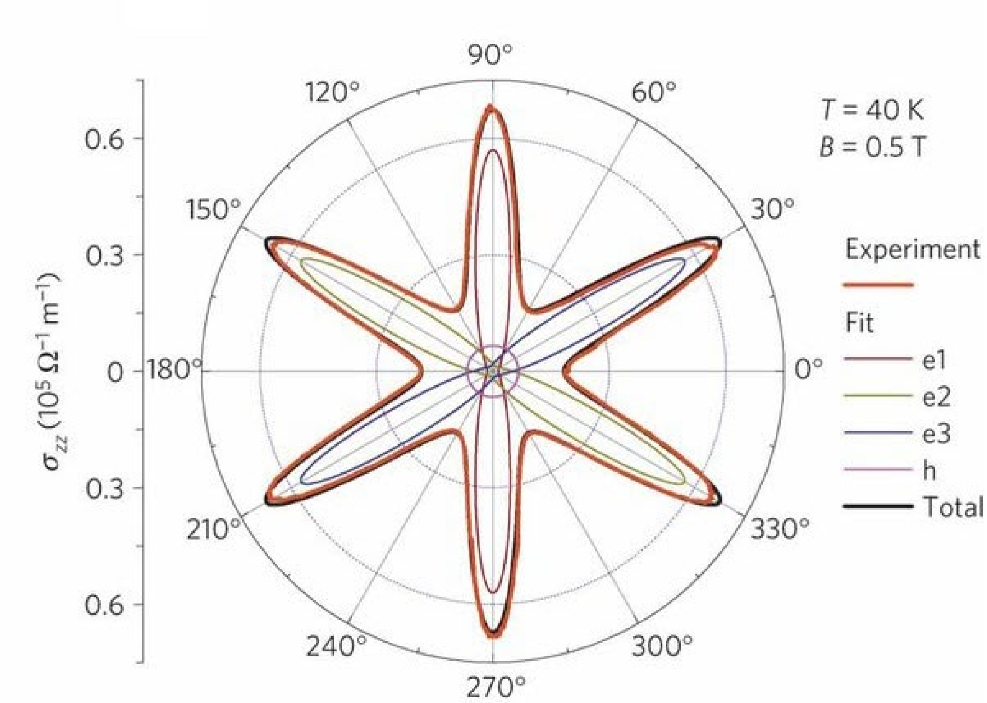
②

$$7 + 7 * \frac{[\sin(16\theta) + \sin(6\theta)]}{2}$$

Even **just two simple waves** when added together can produce complicated results!

Parametric Curves

Field Induced Polarization of Dirac Valleys in Bismuth*



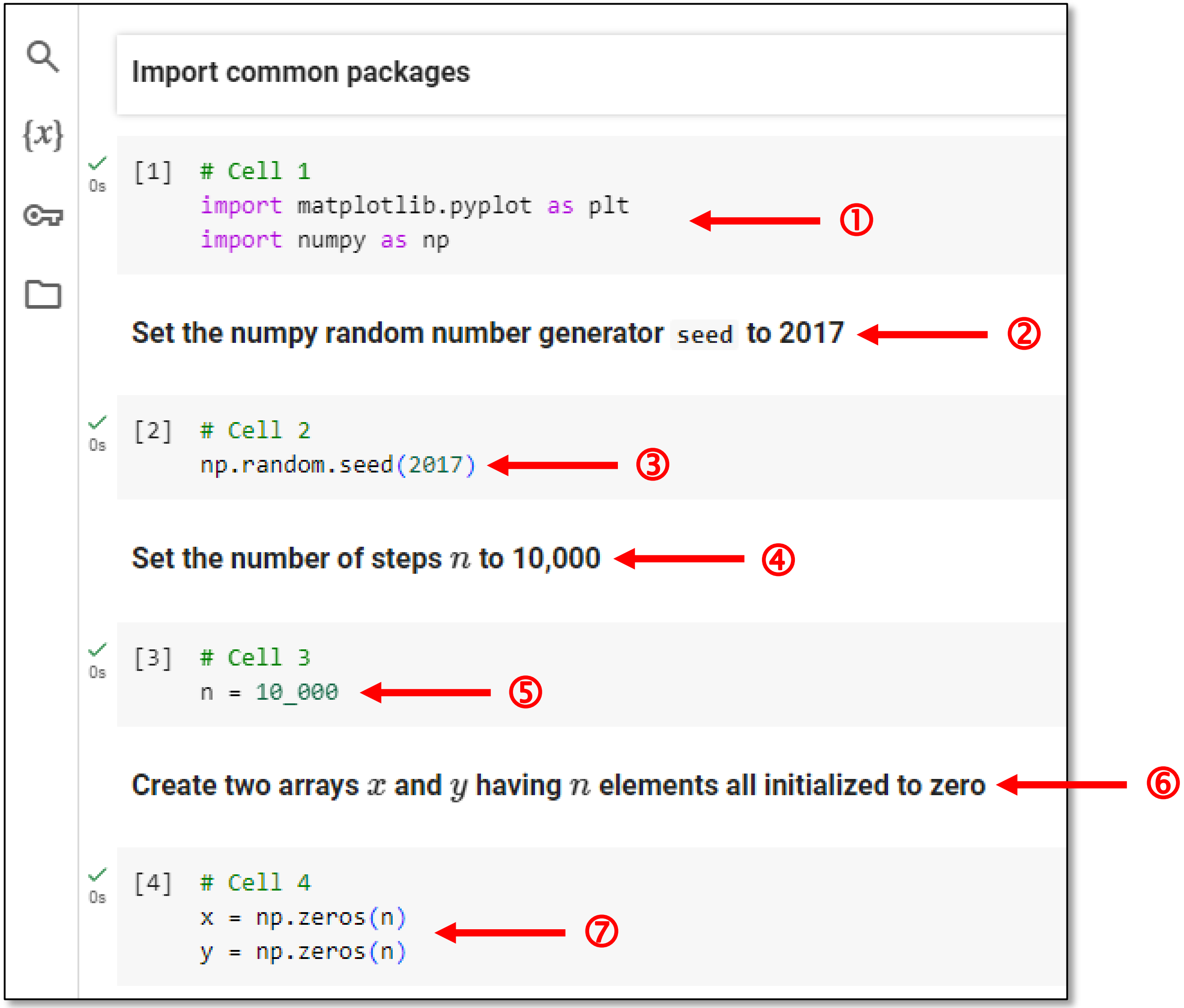
$$r = \sin^2\left(\frac{6}{5}\theta\right) + \cos^2\left(\frac{6}{1}\theta\right)$$

*Bismuth is the element with the **highest** atomic mass that is **stable**

Random Walks

- Your scientist wants you to create a Python program to display the **2D** Cartesian plot of a meandering walker
- The walker starts at the (0,0) origin and takes one step at a time
- At each step, the walker picks a random angle (**uniform** distribution) within the interval $[0,2\pi)$ and moves (from his current position) one unit of distance in that **radial** direction
- Your boss wants your program to show the entire journey of **10,000** random steps in your plot
- On average, how far away (Pythagorean distance) from the start point will the walker **stop**?

Run random_walk.ipynb – Cells 1…4



Import common packages

```
[1]  # Cell 1
     import matplotlib.pyplot as plt
     import numpy as np
```

①

Set the numpy random number generator `seed` to 2017 ②

```
[2]  # Cell 2
     np.random.seed(2017)
```

③

Set the number of steps n to 10,000 ④

```
[3]  # Cell 3
     n = 10_000
```

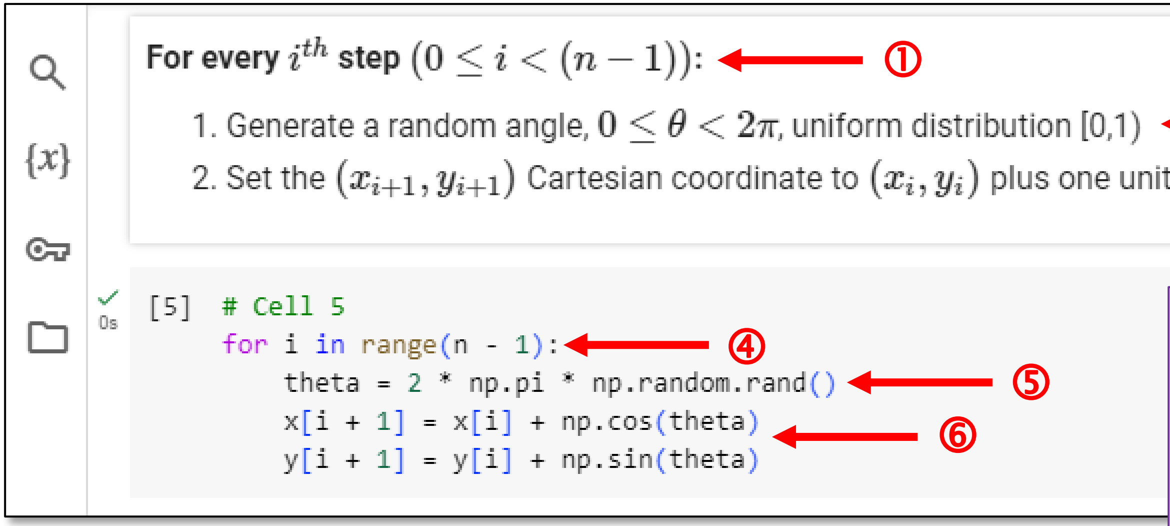
⑤

Create two arrays x and y having n elements all initialized to zero ⑥

```
[4]  # Cell 4
     x = np.zeros(n)
     y = np.zeros(n)
```

⑦

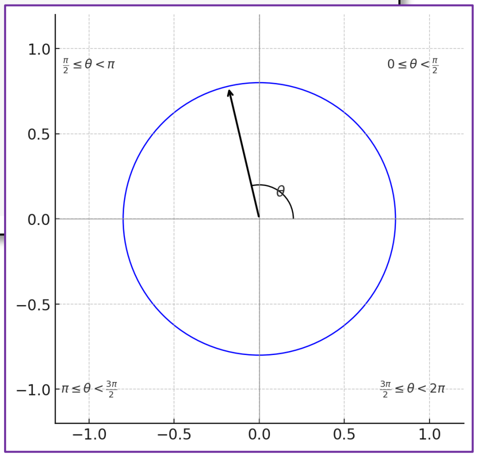
Run random_walk.ipynb – **Cell 5**



For every i^{th} step $(0 \leq i < (n-1))$: ①

1. Generate a random angle, $0 \leq \theta < 2\pi$, uniform distribution [0,1) ②
2. Set the (x_{i+1}, y_{i+1}) Cartesian coordinate to (x_i, y_i) plus one unit step in the θ direction ③

```
[5] # Cell 5
    for i in range(n - 1):                    ④
        theta = 2 * np.pi * np.random.rand()  ⑤
        x[i + 1] = x[i] + np.cos(theta)       ⑥
        y[i + 1] = y[i] + np.sin(theta)
```

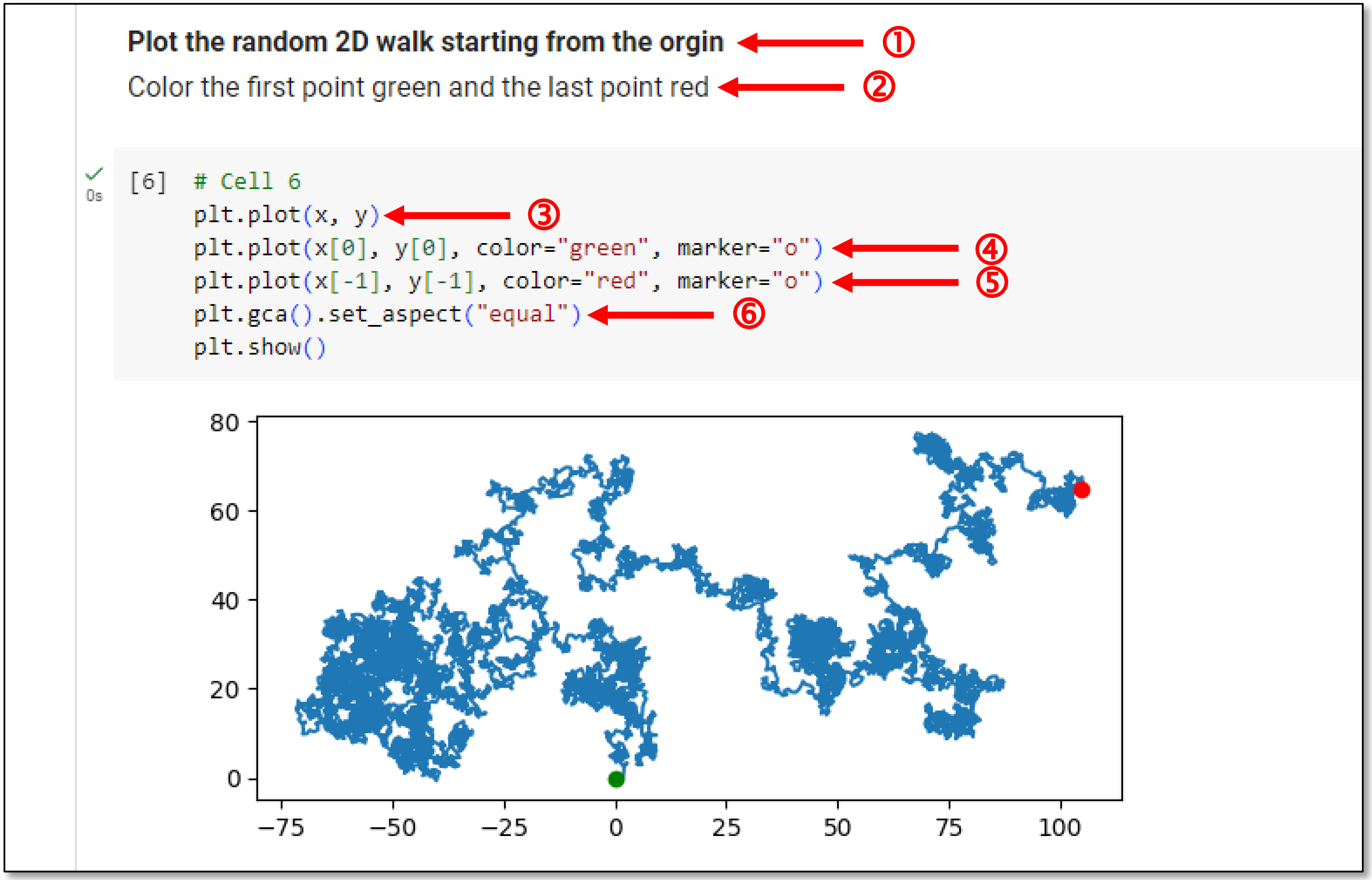

1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
$\frac{\pi}{2} \leq \theta < \pi$
$0 \leq \theta < \frac{\pi}{2}$
$\pi \leq \theta < \frac{3\pi}{2}$
$\frac{3\pi}{2} \leq \theta < 2\pi$
θ

In 2D Cartesian Coordinates:

x[i], y[i] = Where you are currently at

x[i+1], y[i+1] = Where you will be at *after* taking this step

Run random_walk.ipynb – **Cell 6**

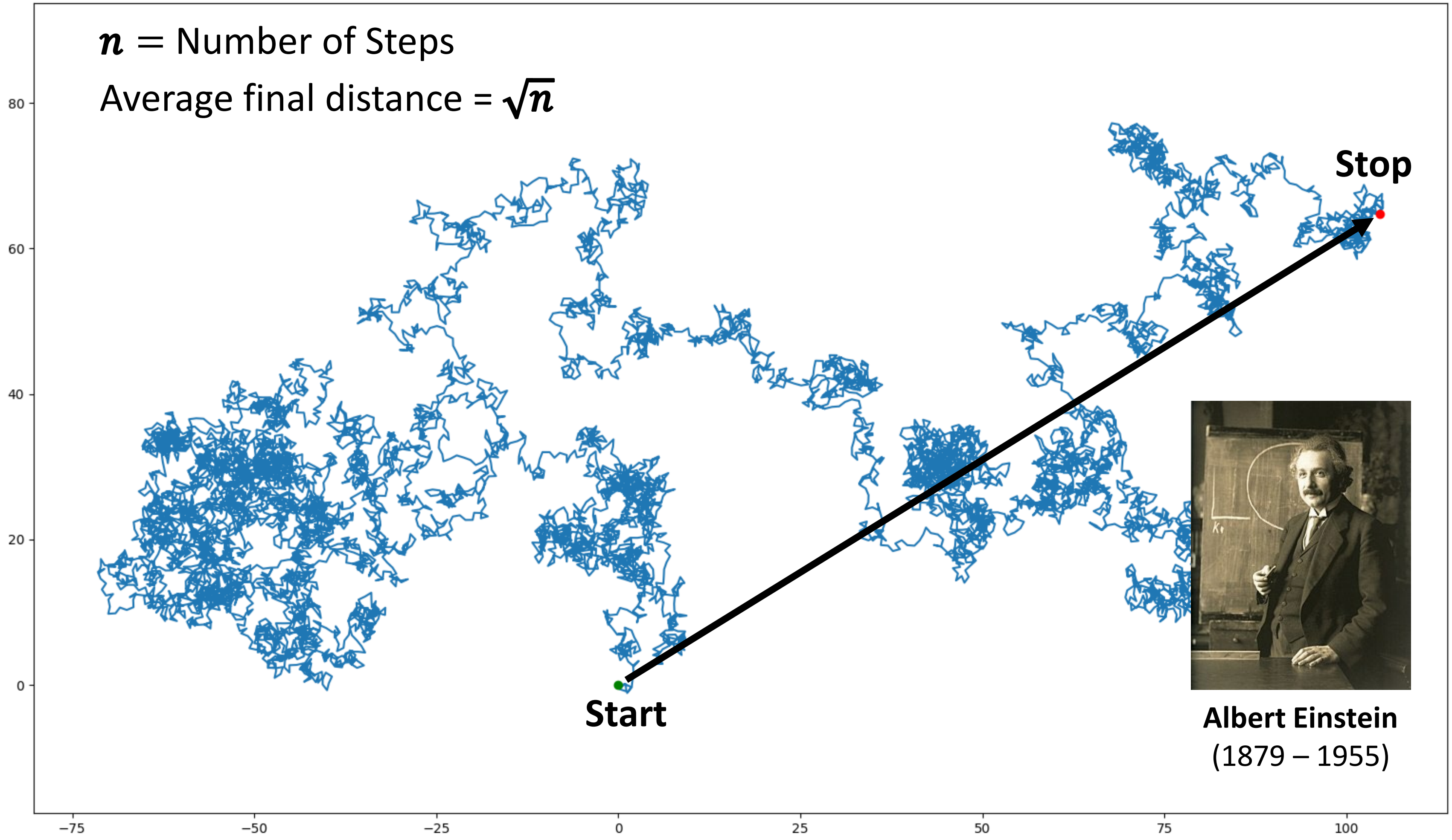


Plot the random 2D walk starting from the orgin

Color the first point green and the last point red

```
# Cell 6
plt.plot(x, y)
plt.plot(x[0], y[0], color="green", marker="o")
plt.plot(x[-1], y[-1], color="red", marker="o")
plt.gca().set_aspect("equal")
plt.show()
```

Random Walks



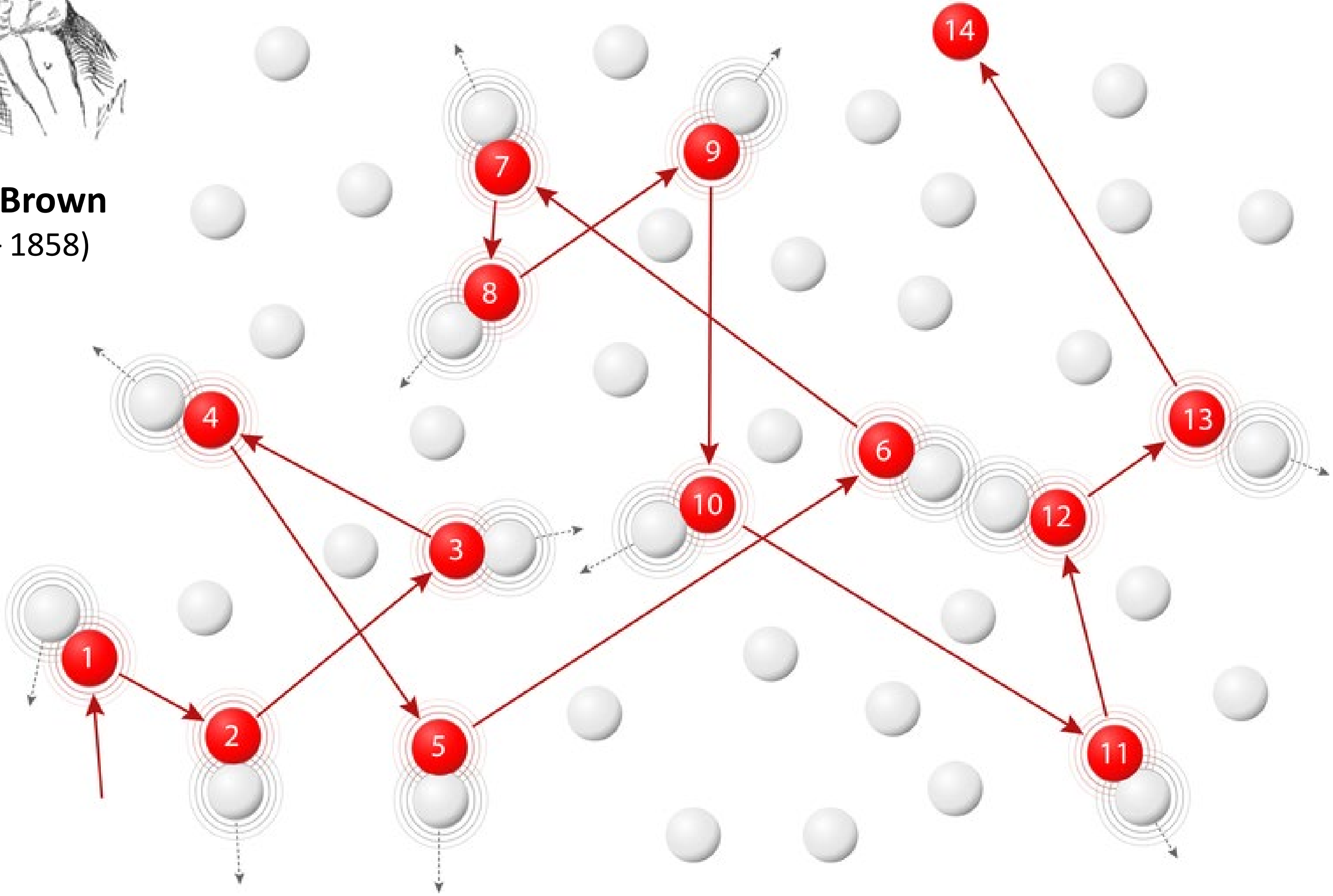
n = Number of Steps

Average final distance = $\sqrt{n}$

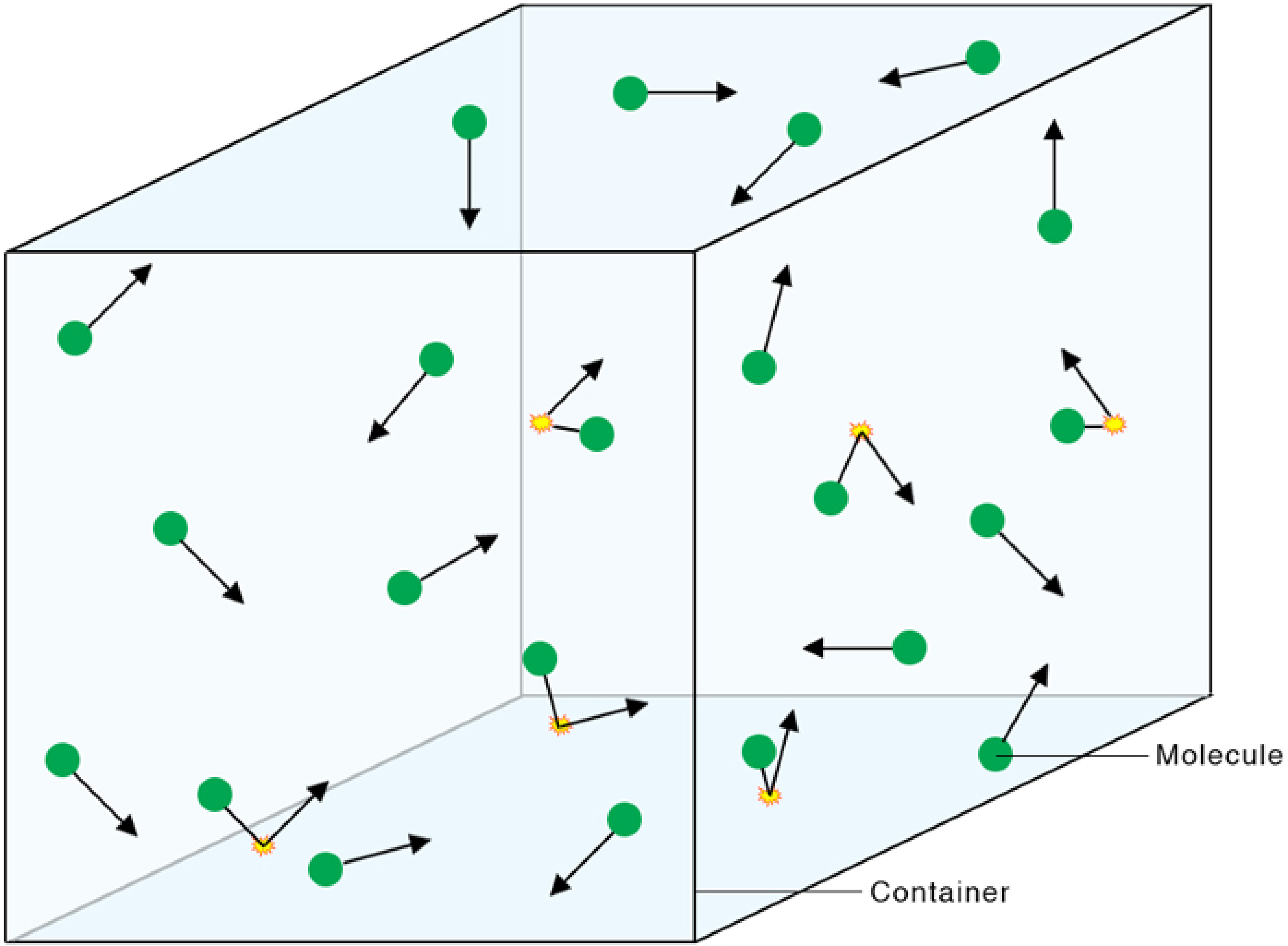
Albert Einstein
(1879 – 1955)

Brownian motion

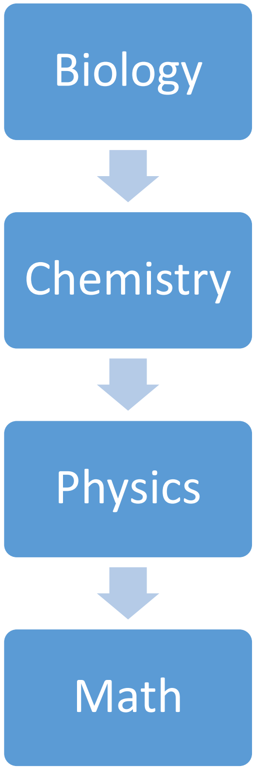
Robert Brown
(1773 – 1858)



Kinetic Theory of Gases



Most of Science is **Waves**



Biology

Chemistry

Physics

Math

- Electrical
- Magnetic
- Acoustic
- Heat Flow
- Vibrational
- Torsional
- Nuclear / Quantum
- Gravitational
- Oceanic / Tidal
- Orbital Precession
- Springs

- Pendulums
- Tomography
- Stock Market
- Economics
- Astronomical
- Fluid Dynamics
- Earthquakes
- AC / DC
- AM / FM
- Speech
- Heartbeats

It is important that you develop a keen understanding of the mathematics of waves!

Traveling Waves & Superposition

$$\lambda = \frac{2\pi}{k} \rightarrow k = \frac{2\pi}{\lambda}$$

$$f = \frac{\omega}{2\pi} \rightarrow \omega = 2\pi f$$

$$y_1 = A_1 \sin(k_1 x + \omega_1 t)$$

$$y_2 = A_2 \sin(k_2 x + \omega_2 t)$$

$$y_1 + y_2 = ?$$

These waves have both **spatial** and **temporal** components

$$y_1 = A_1 \sin(k_1 x + \omega_1 t) = A_1 \sin k_1 x \cos \omega_1 t + A_1 \cos k_1 x \sin \omega_1 t$$

$$y_2 = A_2 \sin(k_2 x + \omega_2 t) = A_2 \sin k_2 x \cos \omega_2 t + A_2 \cos k_2 x \sin \omega_2 t$$

Angle Sum Identity

Simple Case: $A_1 = A_2 = 1, \omega_1 = \omega_2 = 0$

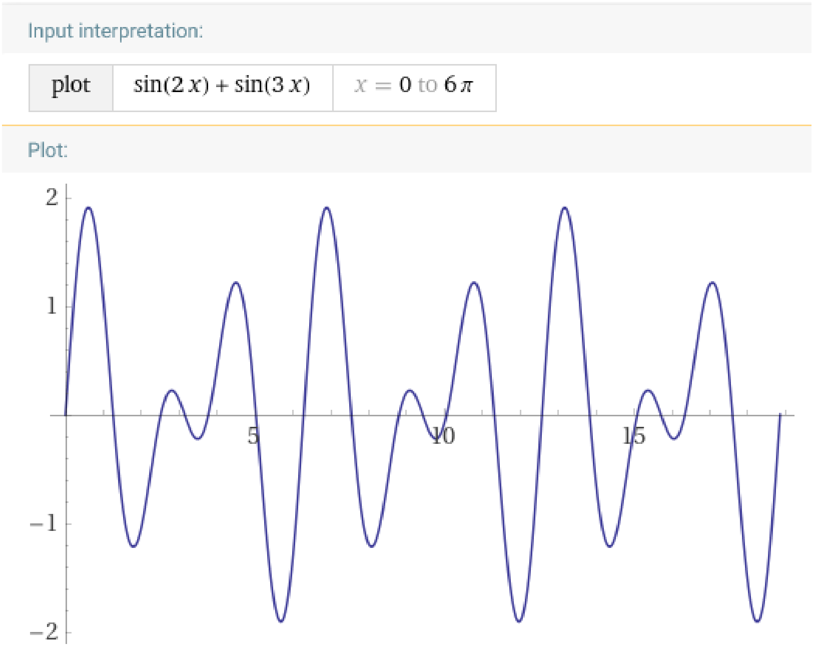
$$y_1 = \sin k_1 x \overset{1}{\cancel{\cos 0t}} + \cancel{\cos k_1 x \sin 0t}$$

$$y_2 = \sin k_2 x \overset{1}{\cancel{\cos 0t}} + \cancel{\cos k_2 x \sin 0t}$$

$$y_1 + y_2 = \sin k_1 x + \sin k_2 x = 2 \sin\left(\frac{(k_1 + k_2)}{2} x\right) \cos\left(\frac{(k_1 - k_2)}{2} x\right)$$

SUM → **PRODUCT**

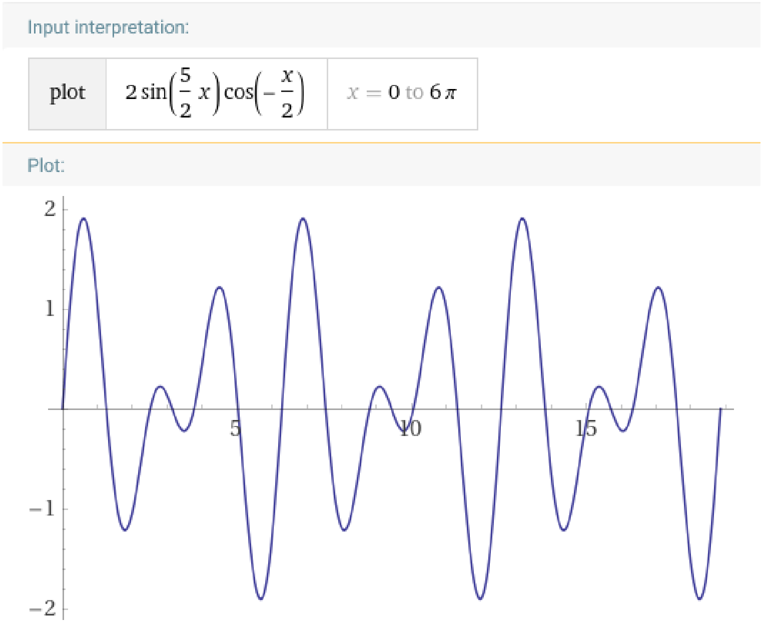
Travelling Waves & Superposition



Input interpretation:

plot | $\sin(2x) + \sin(3x)$ | $x = 0$ to 6π

Plot:



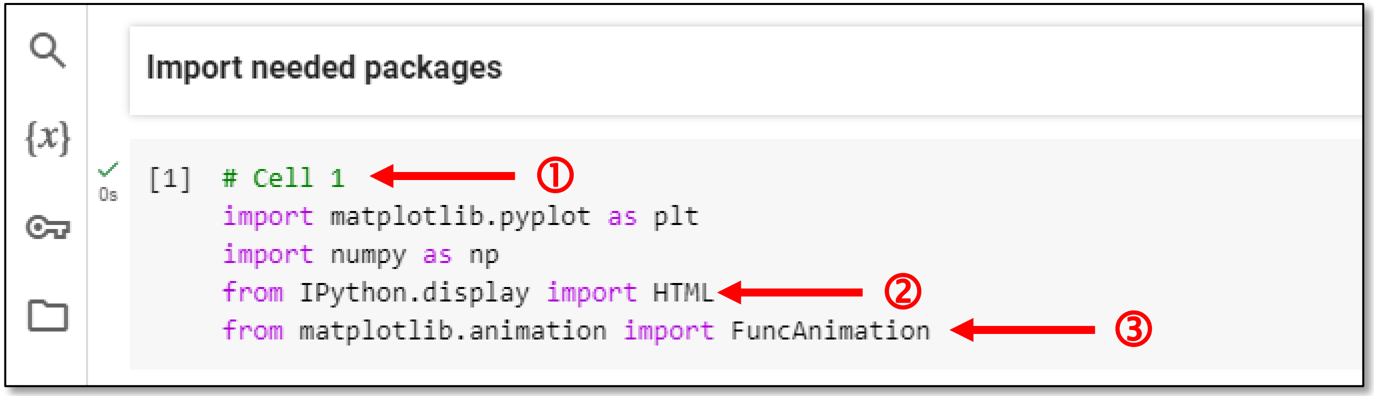
Input interpretation:

plot | $2\sin\left(\frac{5}{2}x\right)\cos\left(-\frac{x}{2}\right)$ | $x = 0$ to 6π

Plot:

But what if the two waves are oscillating at different angular **velocities** or have different **amplitudes**, or different **wave numbers**?

Run traveling_waves.ipynb – **Cell 1**



Import needed packages

```
[1] # Cell 1
    import matplotlib.pyplot as plt
    import numpy as np
    from IPython.display import HTML
    from matplotlib.animation import FuncAnimation
```

Unlock IPython's Magical Toolbox for Your Coding Journey

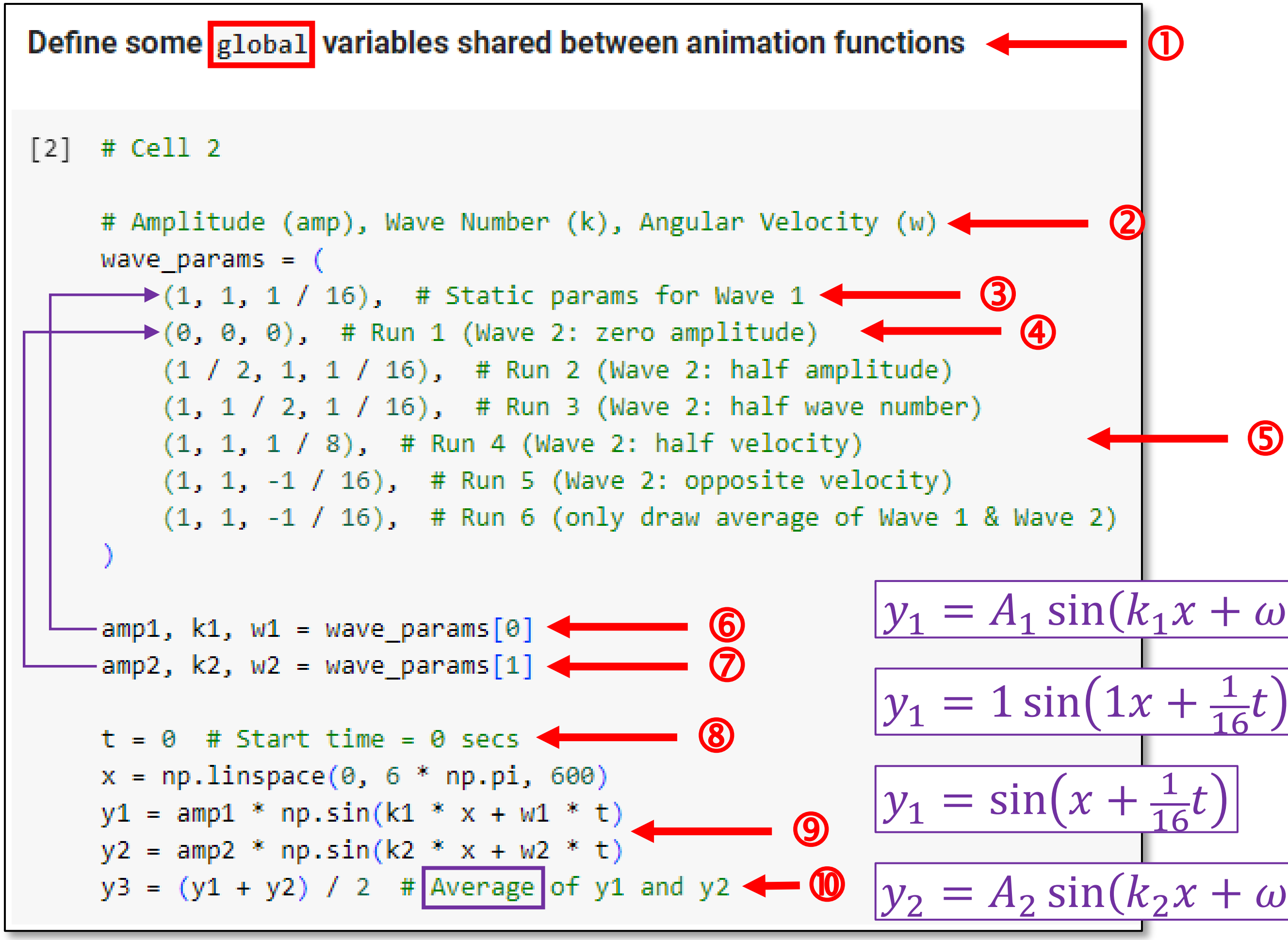
ReadTheDocs: IPython's **Display** Module

Jupyter Notebook: An Introduction

Animations using Matplotlib

Run traveling_waves.ipynb – **Cell 2**

In this notebook, we will only change the parameters of Wave #2



Define some `global` variables shared between animation functions

```
[2]  # Cell 2

     # Amplitude (amp), Wave Number (k), Angular Velocity (w)
     wave_params = (
         (1, 1, 1 / 16),  # Static params for Wave 1
         (0, 0, 0),  # Run 1 (Wave 2: zero amplitude)
         (1 / 2, 1, 1 / 16),  # Run 2 (Wave 2: half amplitude)
         (1, 1 / 2, 1 / 16),  # Run 3 (Wave 2: half wave number)
         (1, 1, 1 / 8),  # Run 4 (Wave 2: half velocity)
         (1, 1, -1 / 16),  # Run 5 (Wave 2: opposite velocity)
         (1, 1, -1 / 16),  # Run 6 (only draw average of Wave 1 & Wave 2)
     )

     amp1, k1, w1 = wave_params[0]
     amp2, k2, w2 = wave_params[1]

     t = 0  # Start time = 0 secs
     x = np.linspace(0, 6 * np.pi, 600)
     y1 = amp1 * np.sin(k1 * x + w1 * t)
     y2 = amp2 * np.sin(k2 * x + w2 * t)
     y3 = (y1 + y2) / 2  # Average of y1 and y2
```

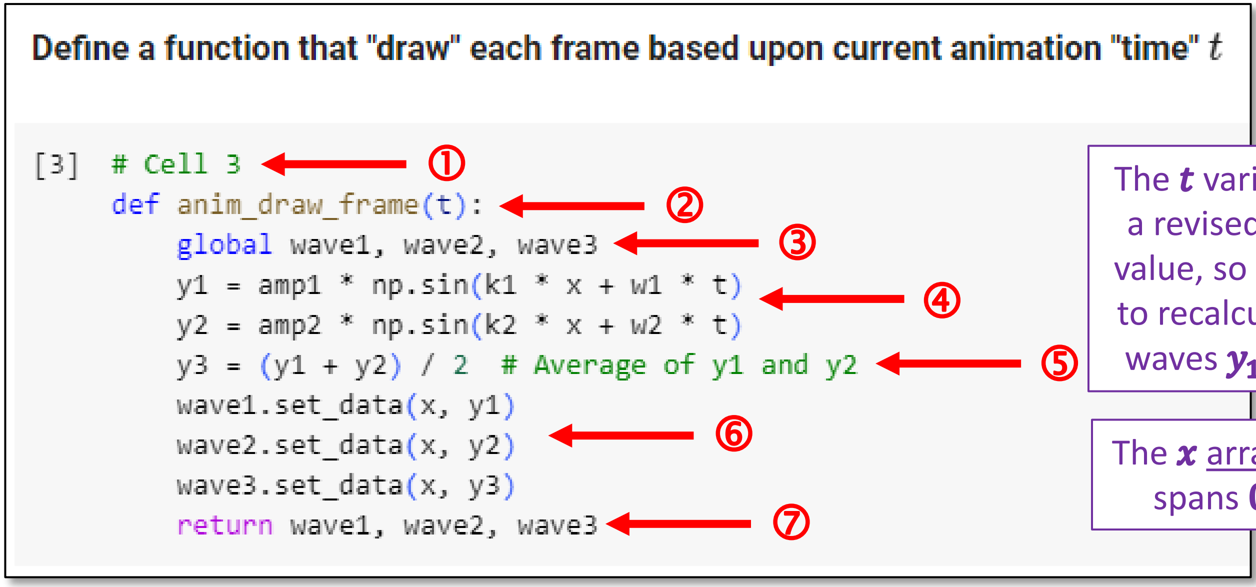
$$\boxed{y_1 = A_1 \sin(k_1 x + \omega_1 t)}$$

$$\boxed{y_1 = 1 \sin\left(1x + \frac{1}{16}t\right)}$$

$$\boxed{y_1 = \sin\left(x + \frac{1}{16}t\right)}$$

$$\boxed{y_2 = A_2 \sin(k_2 x + \omega_2 t)}$$

Run traveling_waves.ipynb – Cell 3



Define a function that "draw" each frame based upon current animation "time" t

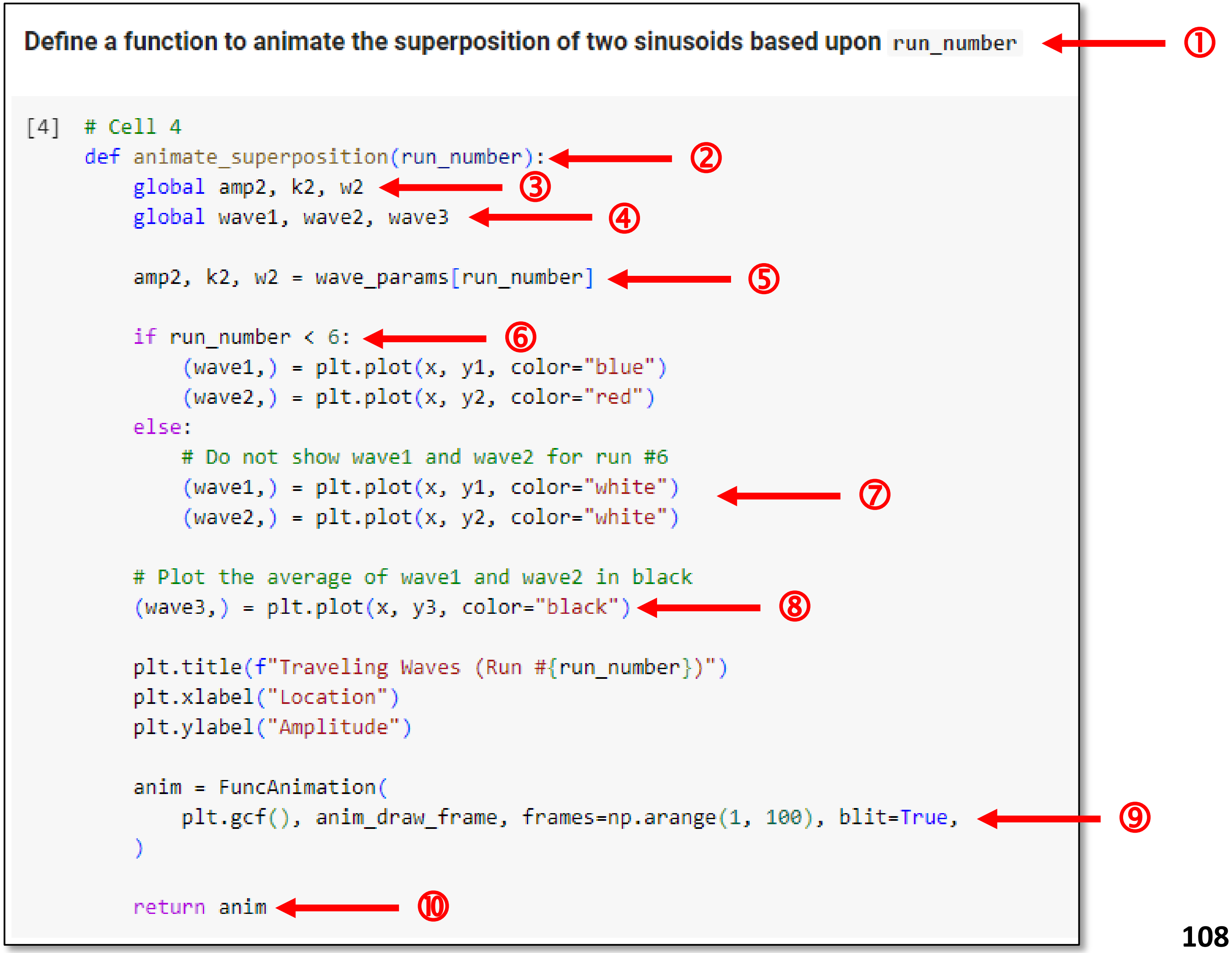
```
[3]  # Cell 3
     def anim_draw_frame(t):
         global wave1, wave2, wave3
         y1 = amp1 * np.sin(k1 * x + w1 * t)
         y2 = amp2 * np.sin(k2 * x + w2 * t)
         y3 = (y1 + y2) / 2  # Average of y1 and y2
         wave1.set_data(x, y1)
         wave2.set_data(x, y2)
         wave3.set_data(x, y3)
         return wave1, wave2, wave3
```

The t variable has a revised “time” value, so we need to recalculate the waves y_1 and y_2

The x array always spans $0 \dots 6\pi$

You *must* use the **global** keyword to specify any global *variables* you intend to modify inside a function

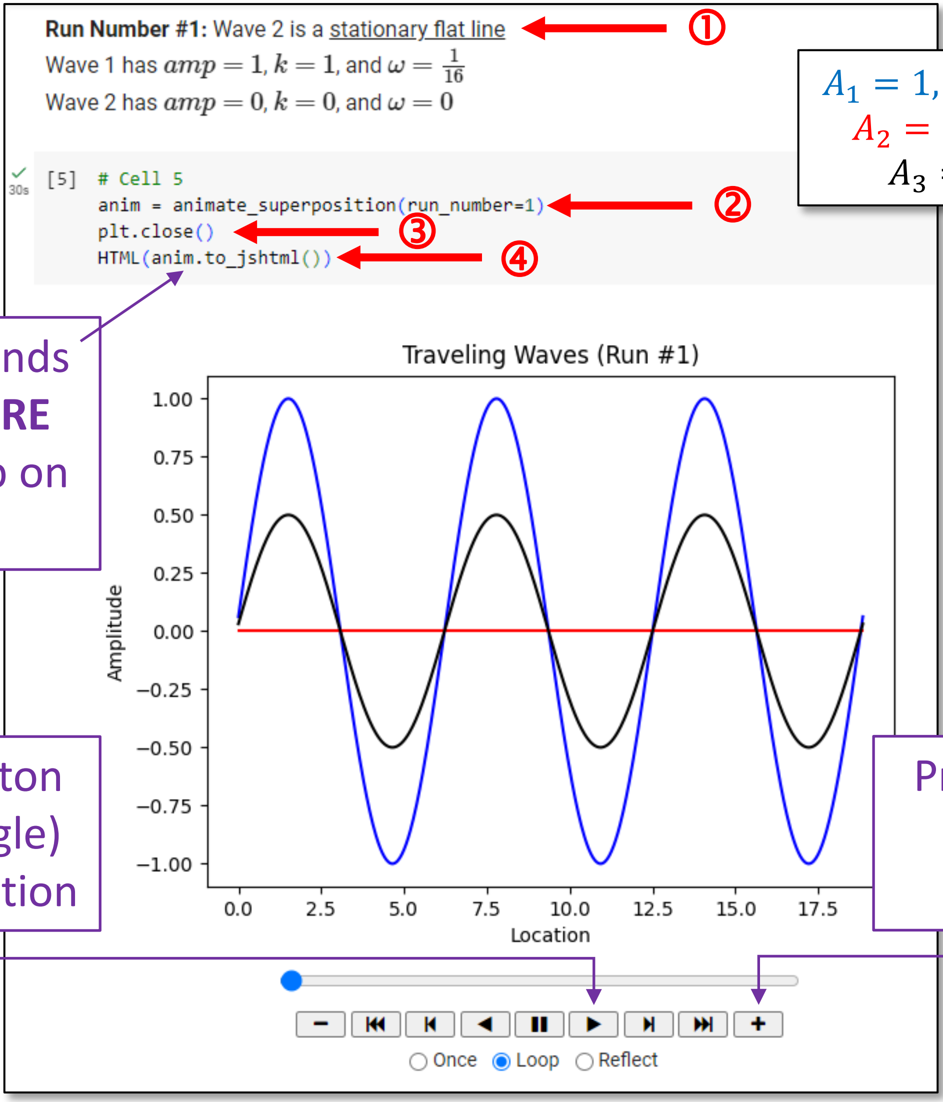
Run traveling_waves.ipynb – **Cell 4**



Define a function to animate the superposition of two sinusoids based upon `run_number`

```
[4]  # Cell 4
     def animate_superposition(run_number):
         global amp2, k2, w2
         global wave1, wave2, wave3

         amp2, k2, w2 = wave_params[run_number]

         if run_number < 6:
             (wave1,) = plt.plot(x, y1, color="blue")
             (wave2,) = plt.plot(x, y2, color="red")
         else:
             # Do not show wave1 and wave2 for run #6
             (wave1,) = plt.plot(x, y1, color="white")
             (wave2,) = plt.plot(x, y2, color="white")

         # Plot the average of wave1 and wave2 in black
         (wave3,) = plt.plot(x, y3, color="black")

         plt.title(f"Traveling Waves (Run #{run_number})")
         plt.xlabel("Location")
         plt.ylabel("Amplitude")

         anim = FuncAnimation(
             plt.gcf(), anim_draw_frame, frames=np.arange(1, 100), blit=True,
         )

         return anim
```

Run traveling_waves.ipynb – **Cell 5**



Run Number #1: Wave 2 is a stationary flat line ①

Wave 1 has $amp = 1, k = 1$, and $\omega = \frac{1}{16}$

Wave 2 has $amp = 0, k = 0$, and $\omega = 0$

✓ 30s [5]

```
# Cell 5
anim = animate_superposition(run_number=1)
plt.close()
HTML(anim.to_jshtml())
```

② ③ ④

○ Once ● Loop ○ Reflect

$A_1 = 1, k_1 = 1, \omega_1 = 1/16$

$A_2 = 0, k_2 = 0, \omega_2 = 0$

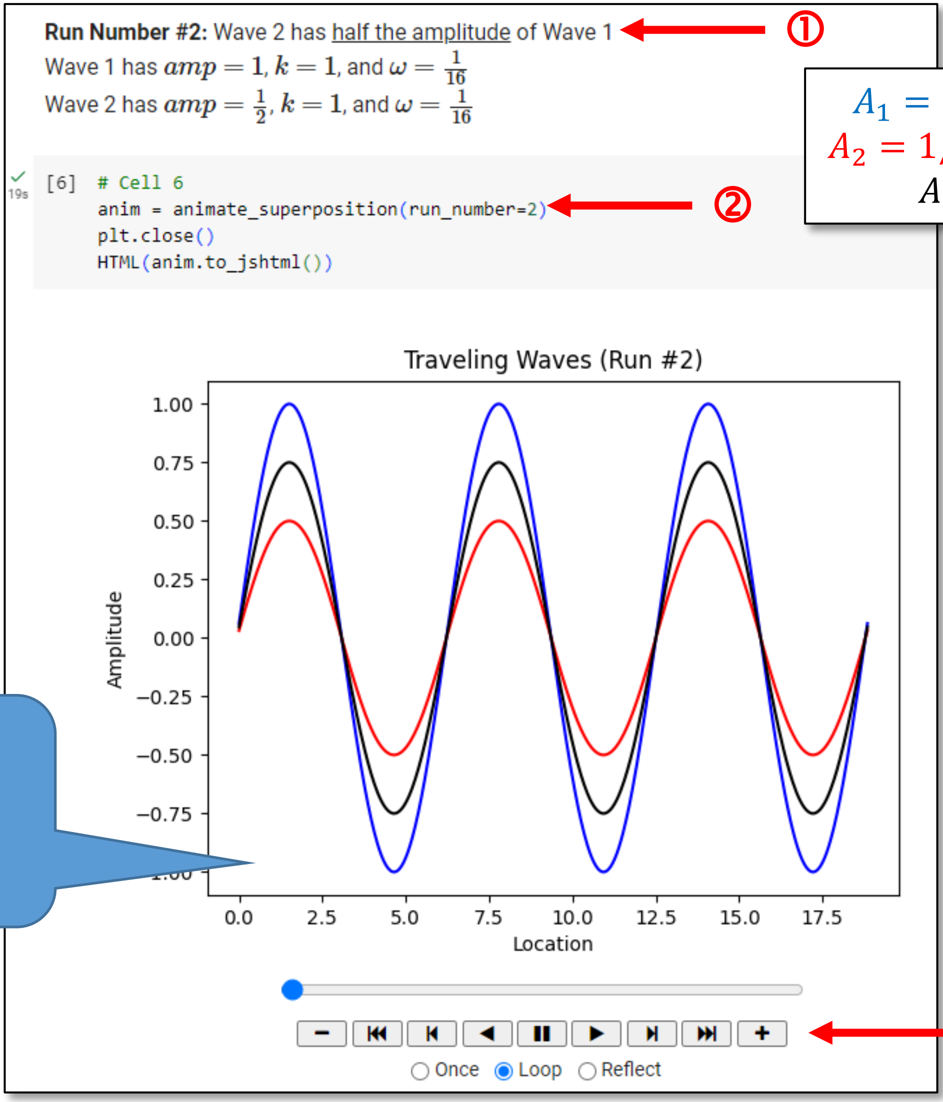
$A_3 = (A_1 + A_2)/2$

It can take **30** seconds to calculate **BEFORE** *anything* shows up on the screen

Press the **play** button (right-facing triangle) to begin the animation

Press the **+** button to **speed up** the animation

Run traveling_waves.ipynb – Cell 6



Run Number #2: Wave 2 has half the amplitude of Wave 1 ①

Wave 1 has $amp = 1, k = 1$, and $\omega = \frac{1}{16}$

Wave 2 has $amp = \frac{1}{2}, k = 1$, and $\omega = \frac{1}{16}$

```
✓   [6]  # Cell 6
19s      anim = animate_superposition(run_number=2)
         plt.close()
         HTML(anim.to_jshtml())
```

②

○ Once ● Loop ○ Reflect

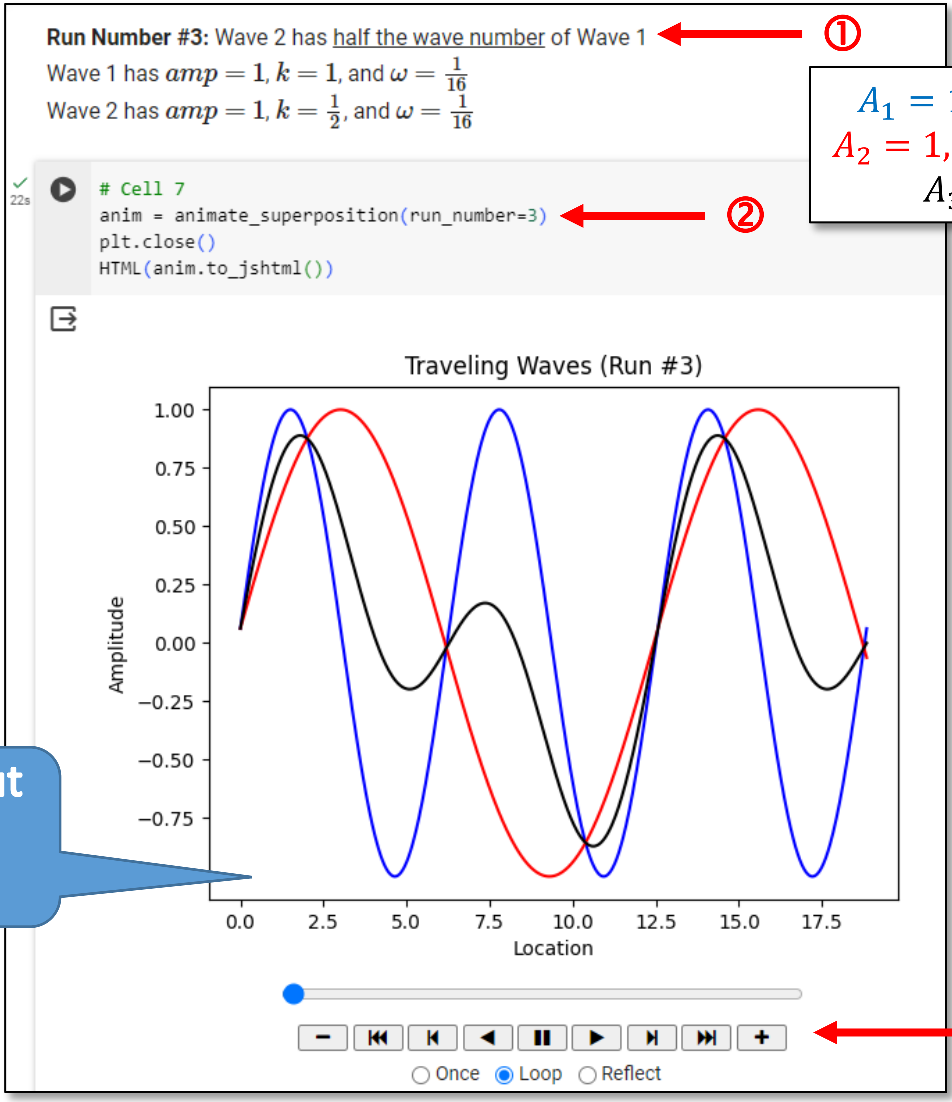
$A_1 = 1, k_1 = 1, \omega_1 = 1/16$

$A_2 = 1/2\,, k_2 = 1, \omega_2 = 1/16$

$A_3 = (A_1 + A_2)/2$

Different amplitudes but same λ and ω

Run traveling_waves.ipynb – Cell 7



Run Number #3: Wave 2 has half the wave number of Wave 1 ①

Wave 1 has $amp = 1, k = 1$, and $\omega = \frac{1}{16}$

Wave 2 has $amp = 1, k = \frac{1}{2}$, and $\omega = \frac{1}{16}$

```
# Cell 7
anim = animate_superposition(run_number=3)
plt.close()
HTML(anim.to_jshtml())
```

②

③

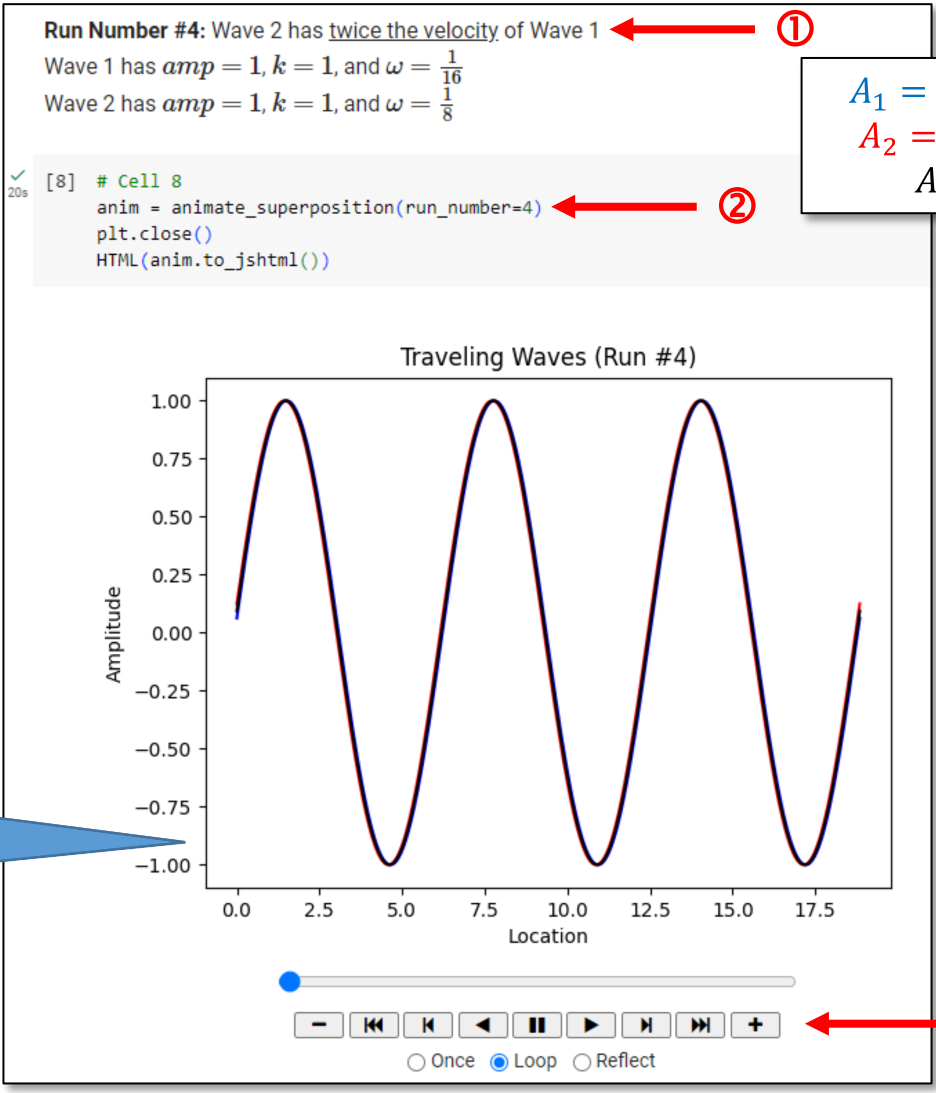
$A_1 = 1, k_1 = 1, \omega_1 = 1/16$

$A_2 = 1, k_2 = 1/2\,, \omega_2 = 1/16$

$A_3 = (A_1 + A_2)/2$

Different λ but same ω and amplitudes

Run traveling_waves.ipynb – **Cell 8**



Run Number #4: Wave 2 has twice the velocity of Wave 1 ①

Wave 1 has $amp = 1$, $k = 1$, and $\omega = \frac{1}{16}$

Wave 2 has $amp = 1$, $k = 1$, and $\omega = \frac{1}{8}$

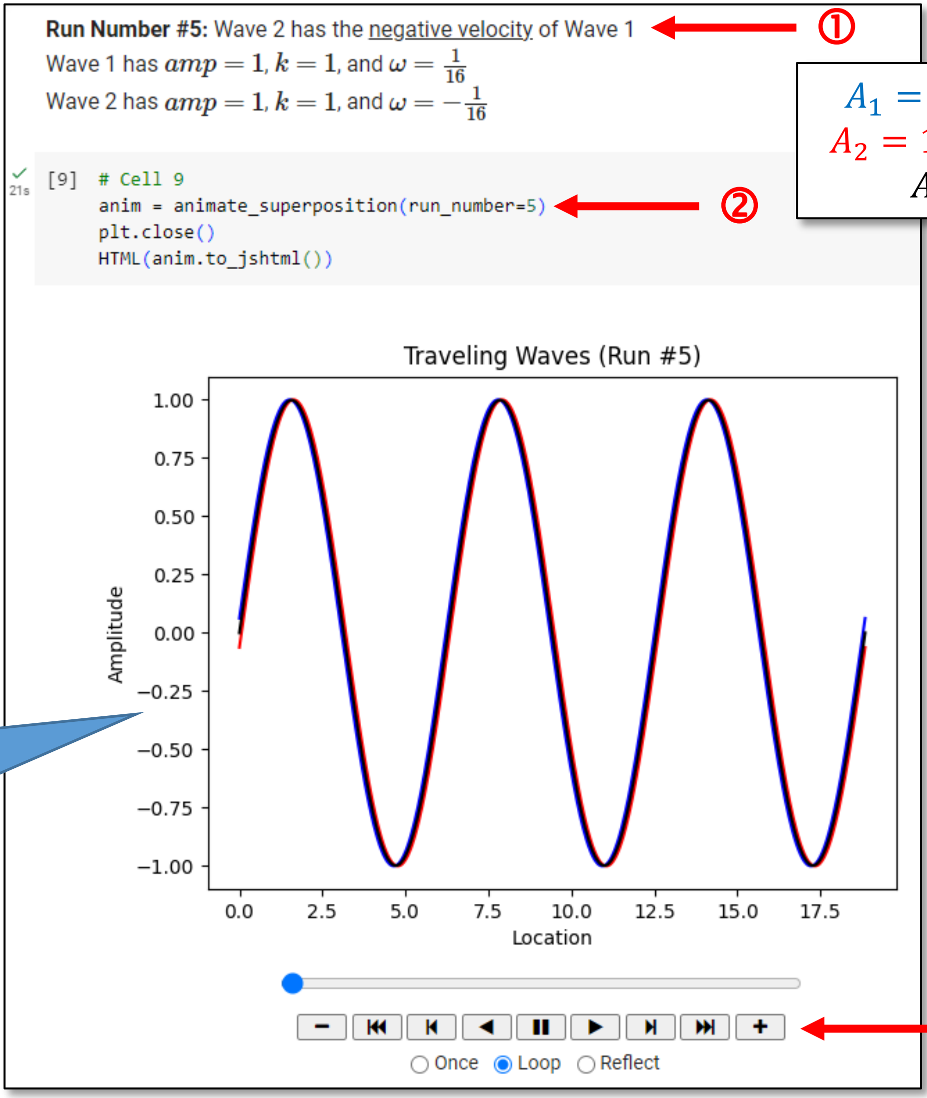
```
[8]  # Cell 8
     anim = animate_superposition(run_number=4)    ②
     plt.close()
     HTML(anim.to_jshtml())
```

○ Once ● Loop ○ Reflect

$$A_1 = 1, k_1 = 1, \omega_1 = 1/16$$
$$A_2 = 1, k_2 = 1, \omega_2 = 1/8$$
$$A_3 = (A_1 + A_2)/2$$

Same amplitudes and λ but different ω

Run traveling_waves.ipynb – **Cell 9**



Run Number #5: Wave 2 has the negative velocity of Wave 1 ← ①

Wave 1 has $amp = 1, k = 1$, and $\omega = \frac{1}{16}$

Wave 2 has $amp = 1, k = 1$, and $\omega = -\frac{1}{16}$

```
[9]  # Cell 9
     anim = animate_superposition(run_number=5)   ← ②
     plt.close()
     HTML(anim.to_jshtml())
```

○ Once ● Loop ○ Reflect

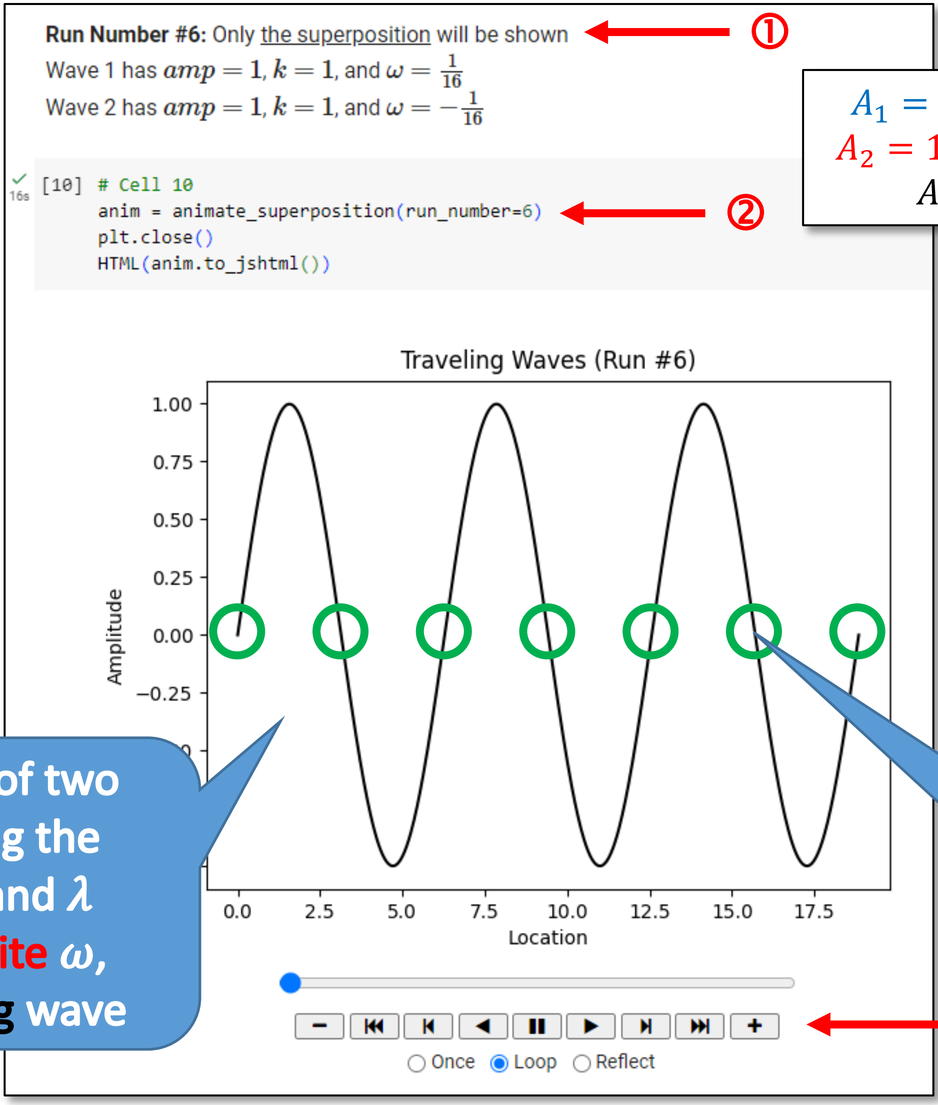
$A_1 = 1, k_1 = 1, \omega_1 = 1/16$

$A_2 = 1, k_2 = 1, \omega_2 = -1/16$

$A_3 = (A_1 + A_2)/2$

Same λ and amplitudes but opposite ω

Run traveling_waves.ipynb – **Cell 10**



Run Number #6: Only the superposition will be shown ①

Wave 1 has $amp = 1$, $k = 1$, and $\omega = \frac{1}{16}$

Wave 2 has $amp = 1$, $k = 1$, and $\omega = -\frac{1}{16}$

```
[10] # Cell 10
     anim = animate_superposition(run_number=6)
     plt.close()
     HTML(anim.to_jshtml())
```

②

③

$A_1 = 1, k_1 = 1, \omega_1 = 1/16$

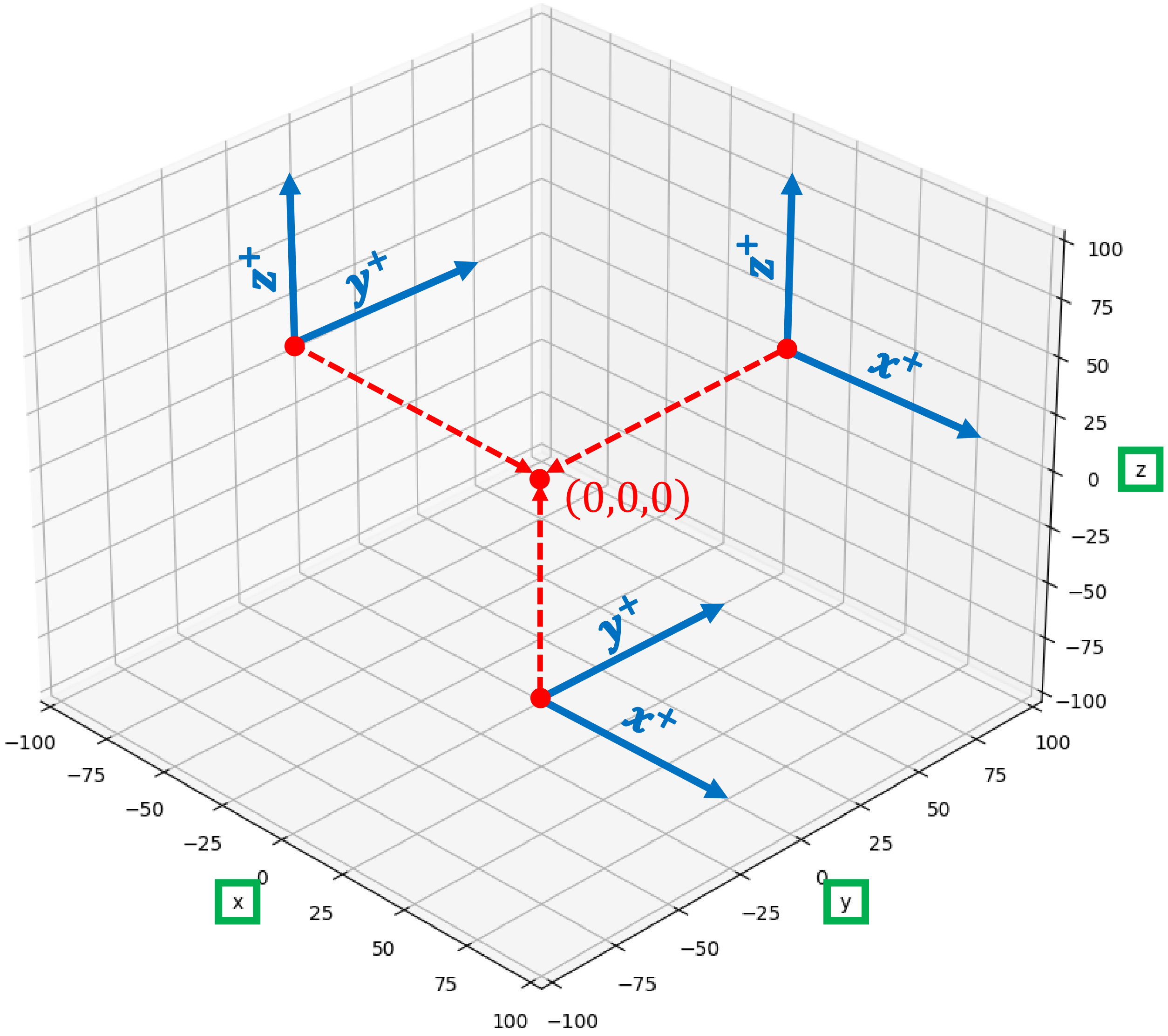
$A_2 = 1, k_2 = 1, \omega_2 = -1/16$

$A_3 = (A_1 + A_2)/2$

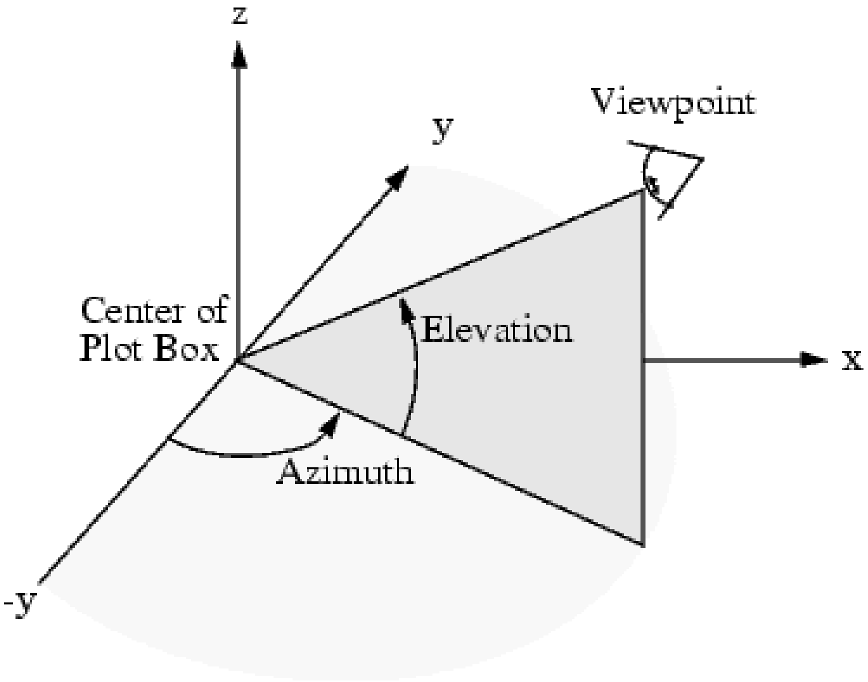
The superposition of two waves, each having the same amplitude and λ but with an opposite ω, produces a standing wave

What points are at the *center* of these circles?

3D Cartesian Coordinates in matplotlib



Viewing Angles in matplotlib



Default view angles:

$\mathbf{azim} = -60^{\circ}$

$\mathbf{elev} = \ 30^{\circ}$

Poloidal and Toroidal Angles

$0 \le u \le \pi \Longrightarrow$ **poloidal** (latitude)
North to South Pole (vertical)

$R =$ radius of sphere

$z = R\cos u$

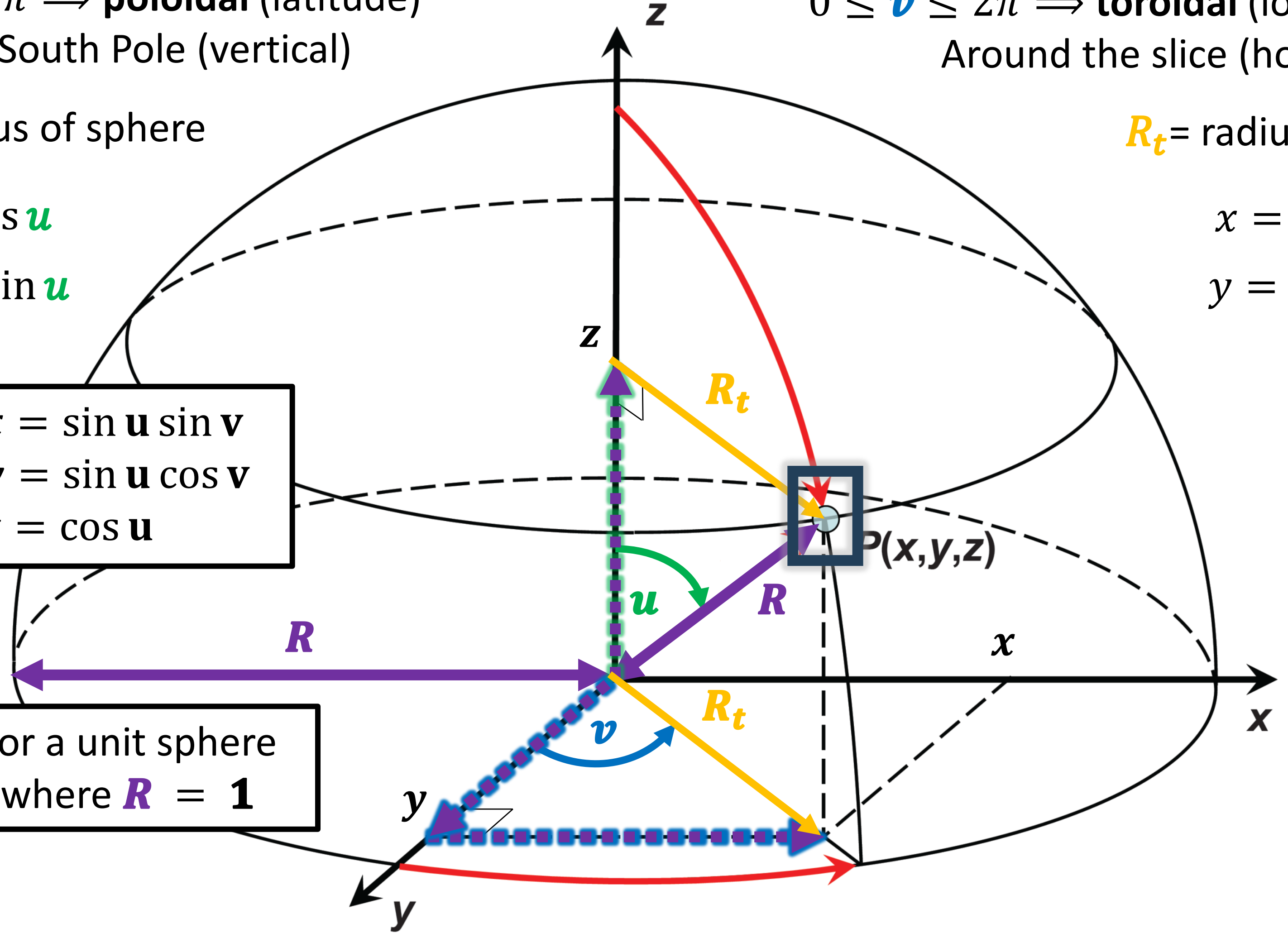
$R_t = R\sin u$

$0 \le v \le 2\pi \Longrightarrow$ **toroidal** (longitude)
Around the slice (horizontal)

R_t= radius of slice

$x = R_t \sin v$

$y = R_t \cos v$



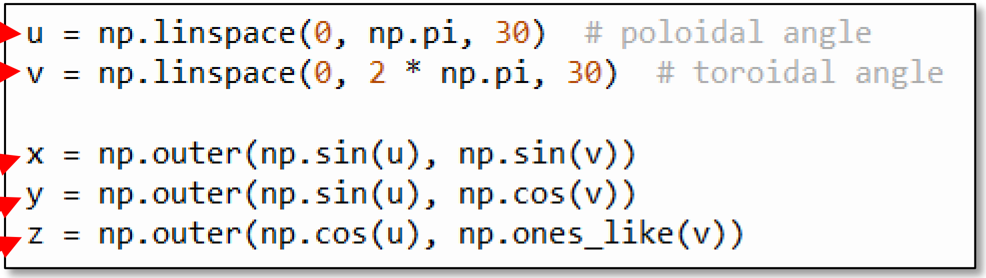
$$x = \sin \mathbf{u} \sin \mathbf{v}$$
$$y = \sin \mathbf{u} \cos \mathbf{v}$$
$$z = \cos \mathbf{u}$$

For a unit sphere
where $R = 1$

Matrices and Outer Product

$$\mathbf{u} = (u_1, u_2, \cdots, u_m)$$

$$\mathbf{v} = (v_1, v_2, \cdots, v_n)$$



```
u = np.linspace(0, np.pi, 30)  # poloidal angle
v = np.linspace(0, 2 * np.pi, 30)  # toroidal angle

x = np.outer(np.sin(u), np.sin(v))
y = np.outer(np.sin(u), np.cos(v))
z = np.outer(np.cos(u), np.ones_like(v))
```

$$x = \sin\mathbf{u} \otimes \sin\mathbf{v}$$

$$y = \sin\mathbf{u} \otimes \cos\mathbf{v}$$

$$z = \cos\mathbf{u} \otimes \mathbf{1}$$

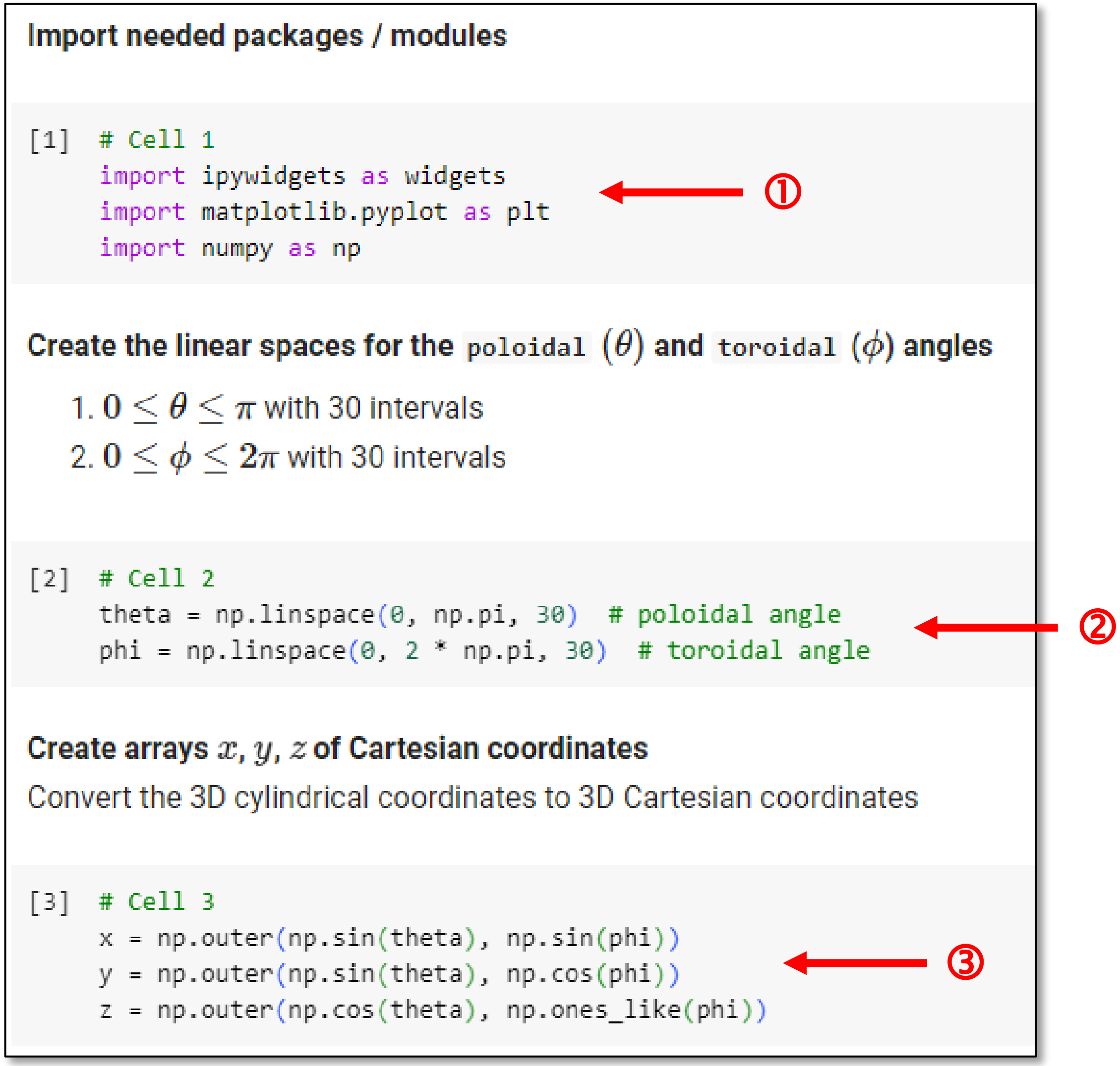
For a unit sphere where $\boldsymbol{R} = \mathbf{1}$

$$x = \sin\mathbf{u}\sin\mathbf{v}$$
$$y = \sin\mathbf{u}\cos\mathbf{v}$$
$$z = \cos\mathbf{u}$$

$$\mathbf{a}\otimes\mathbf{b} = \begin{bmatrix} a_1b_1 & a_1b_2 & \cdots & ab_n \\ a_2b_1 & a_2b_2 & & a_2b_n \\ \vdots & & \ddots & \vdots \\ a_mb_1 & a_mb_2 & \cdots & a_mb_n \end{bmatrix}$$

Outer Product of Two Vectors

Run plot3d_sphere.ipynb – Cells 1..3



Import needed packages / modules

```
[1]  # Cell 1
     import ipywidgets as widgets
     import matplotlib.pyplot as plt
     import numpy as np
```

①

Create the linear spaces for the `poloidal` (θ) and `toroidal` (ϕ) angles

1. $0 \leq \theta \leq \pi$ with 30 intervals
2. $0 \leq \phi \leq 2\pi$ with 30 intervals

```
[2]  # Cell 2
     theta = np.linspace(0, np.pi, 30)  # poloidal angle
     phi = np.linspace(0, 2 * np.pi, 30)  # toroidal angle
```

②

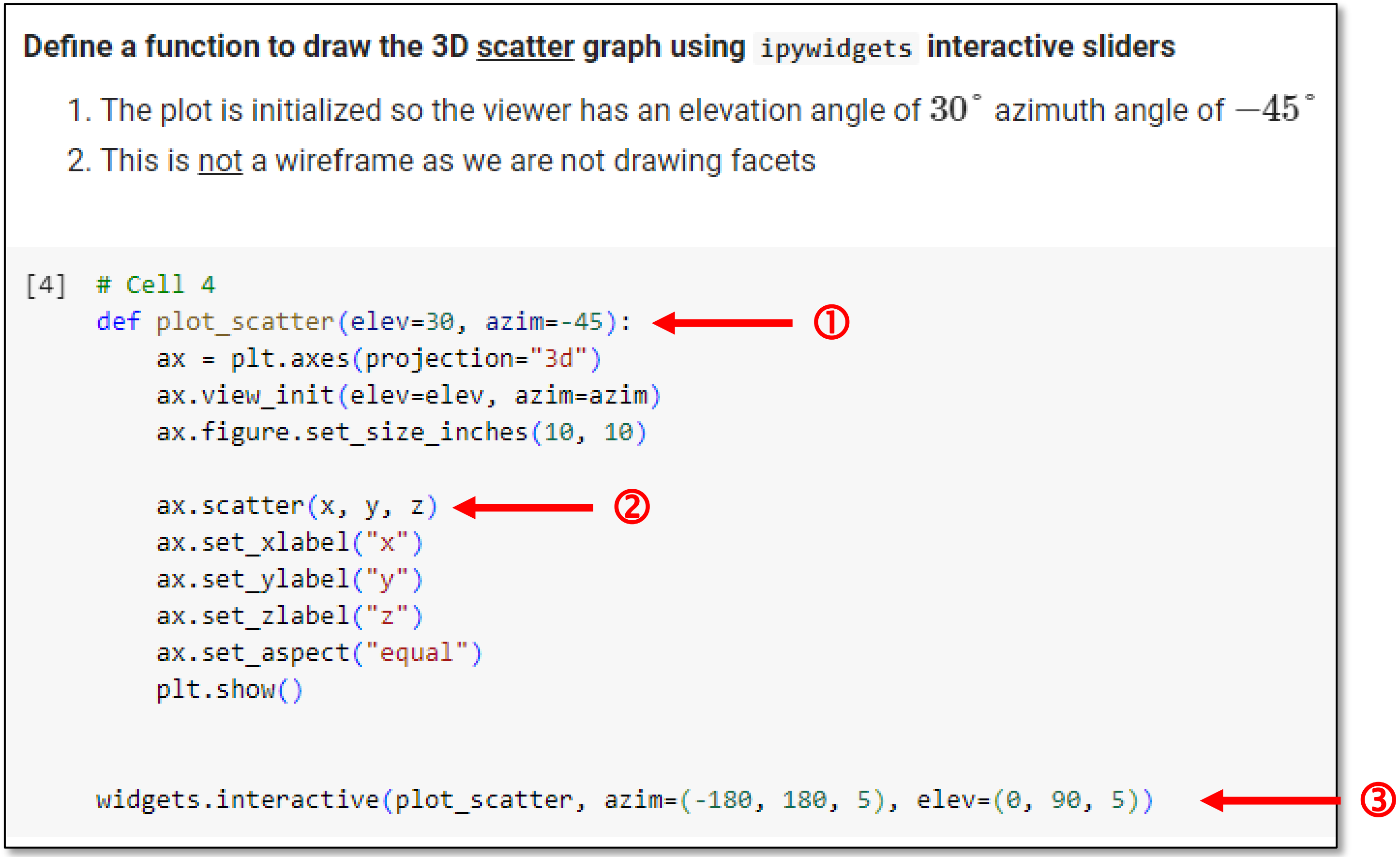
Create arrays x, y, z of Cartesian coordinates

Convert the 3D cylindrical coordinates to 3D Cartesian coordinates

```
[3]  # Cell 3
     x = np.outer(np.sin(theta), np.sin(phi))
     y = np.outer(np.sin(theta), np.cos(phi))
     z = np.outer(np.cos(theta), np.ones_like(phi))
```

③

Run plot3d_sphere.ipynb – **Cell 4**

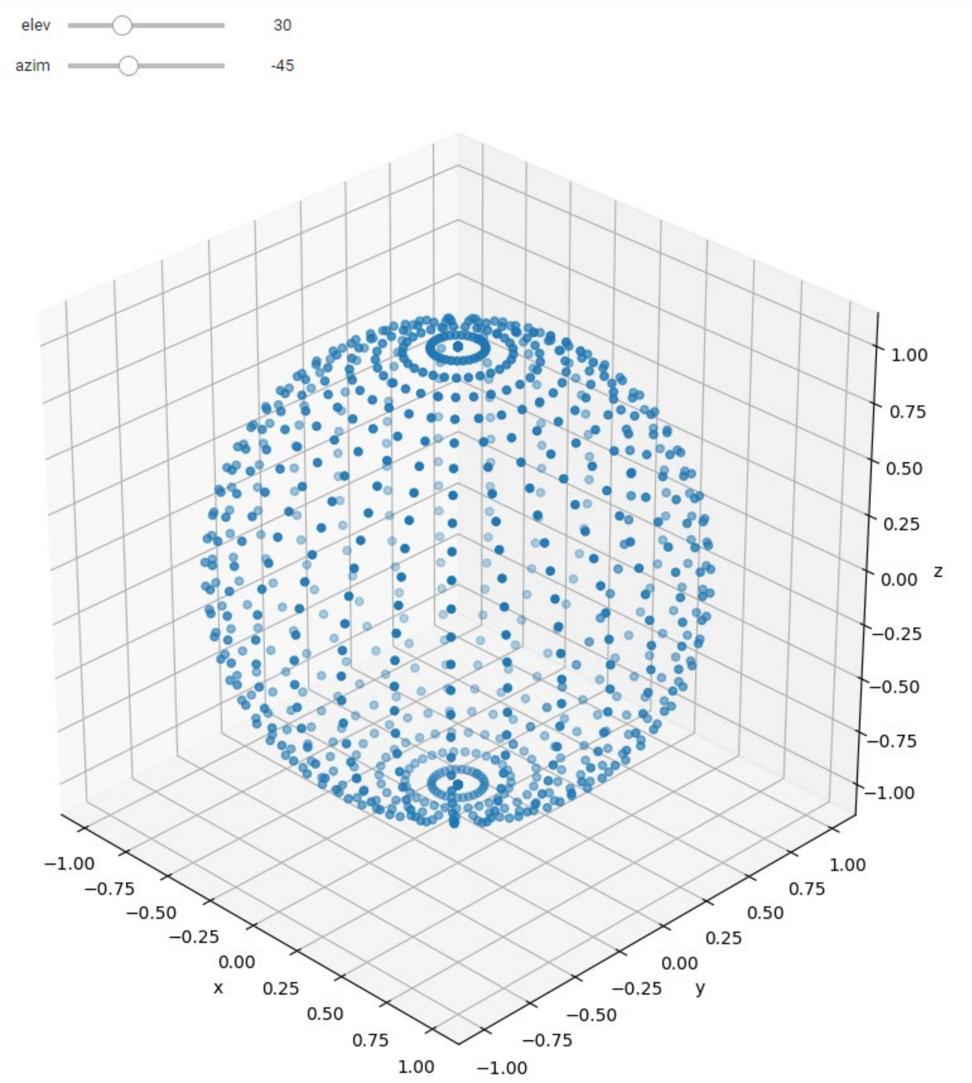


Define a function to draw the 3D scatter graph using `ipywidgets` interactive sliders

1. The plot is initialized so the viewer has an elevation angle of 30° azimuth angle of -45°
2. This is not a wireframe as we are not drawing facets

```
[4]  # Cell 4
     def plot_scatter(elev=30, azim=-45):
         ax = plt.axes(projection="3d")
         ax.view_init(elev=elev, azim=azim)
         ax.figure.set_size_inches(10, 10)

         ax.scatter(x, y, z)
         ax.set_xlabel("x")
         ax.set_ylabel("y")
         ax.set_zlabel("z")
         ax.set_aspect("equal")
         plt.show()


     widgets.interactive(plot_scatter, azim=(-180, 180, 5), elev=(0, 90, 5))
```

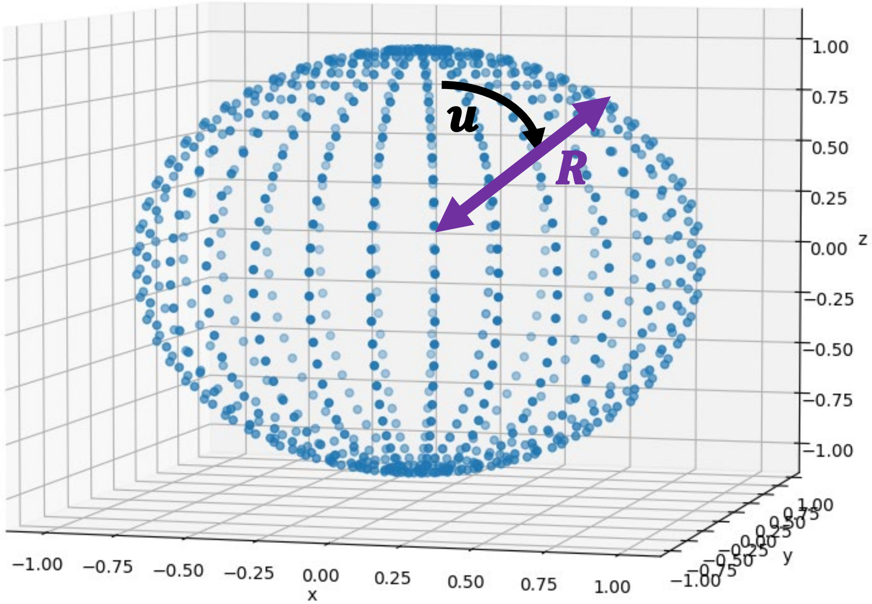
Run plot3d_sphere.ipynb – **Cell 4**



elev 30

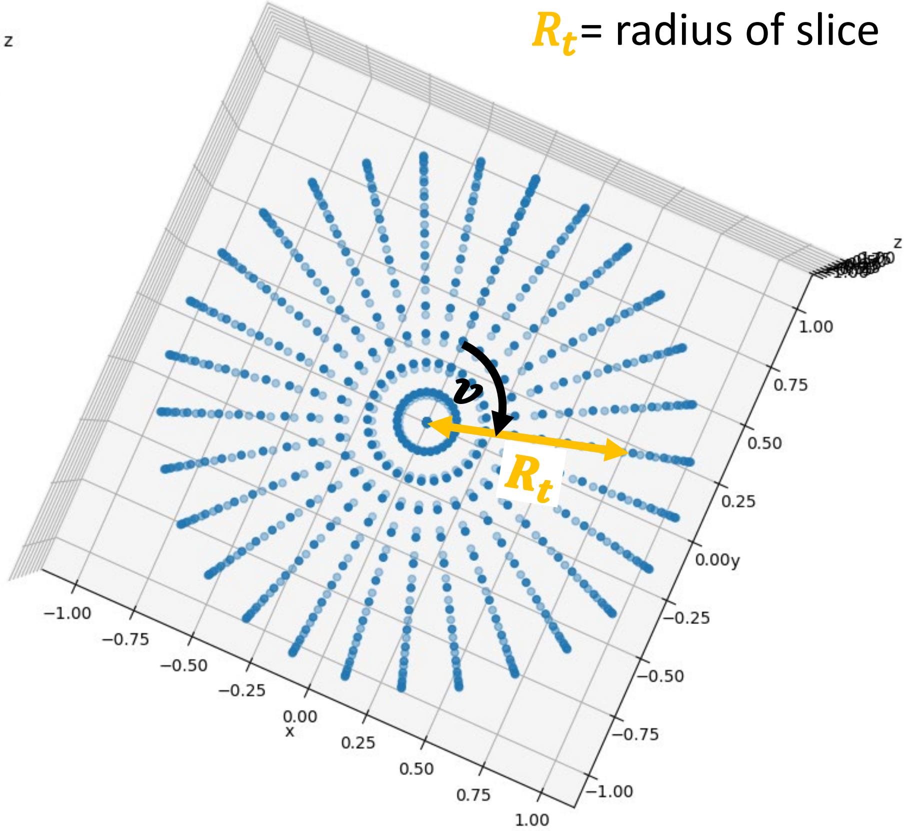
azim -45

Spherical Coordinates



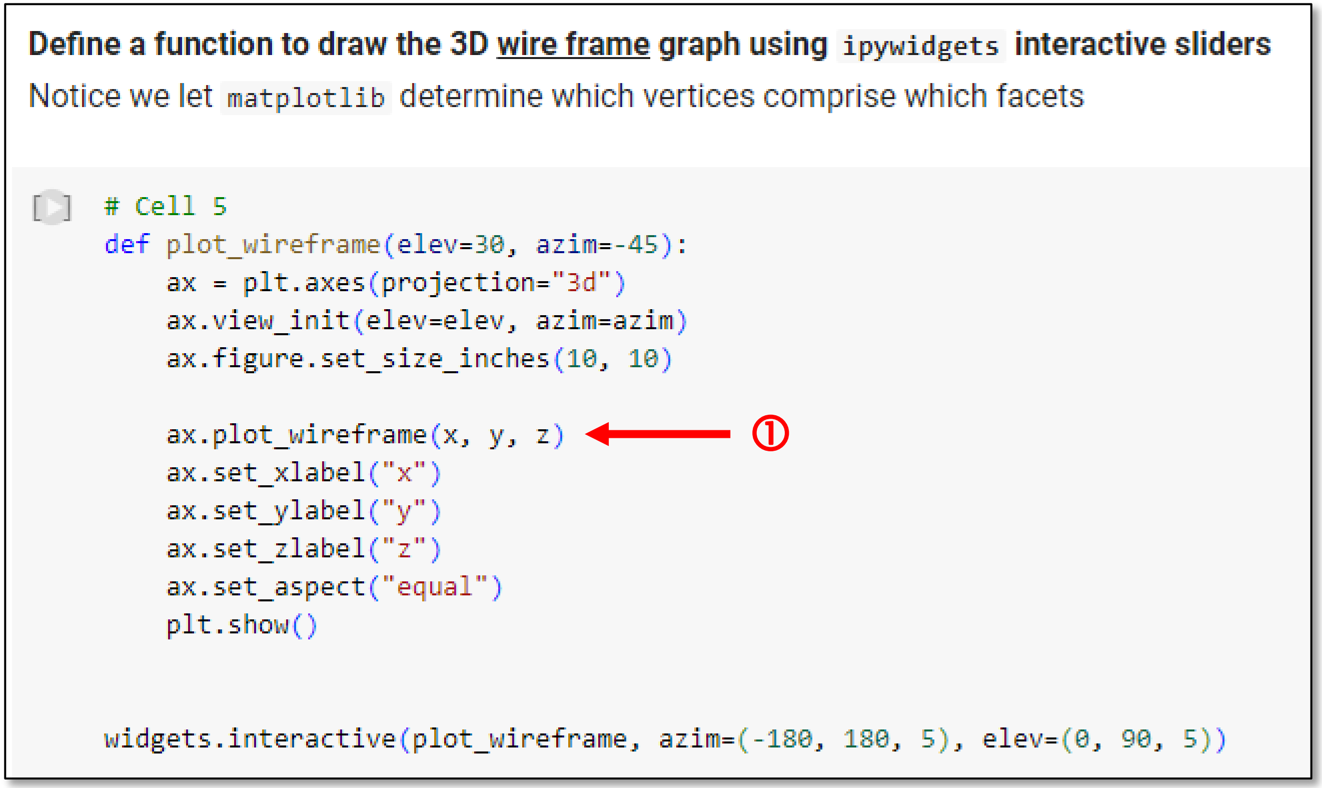
$0 \leq \boldsymbol{v} \leq 2\pi \Longrightarrow$ **toroidal** (longitude)
Around the slice (horizontal)

R_t= radius of slice



$0 \leq \boldsymbol{u} \leq \pi \Longrightarrow$ **poloidal** (latitude)
North to South Pole (vertical)

Run plot3d_sphere.ipynb – **Cell 5**

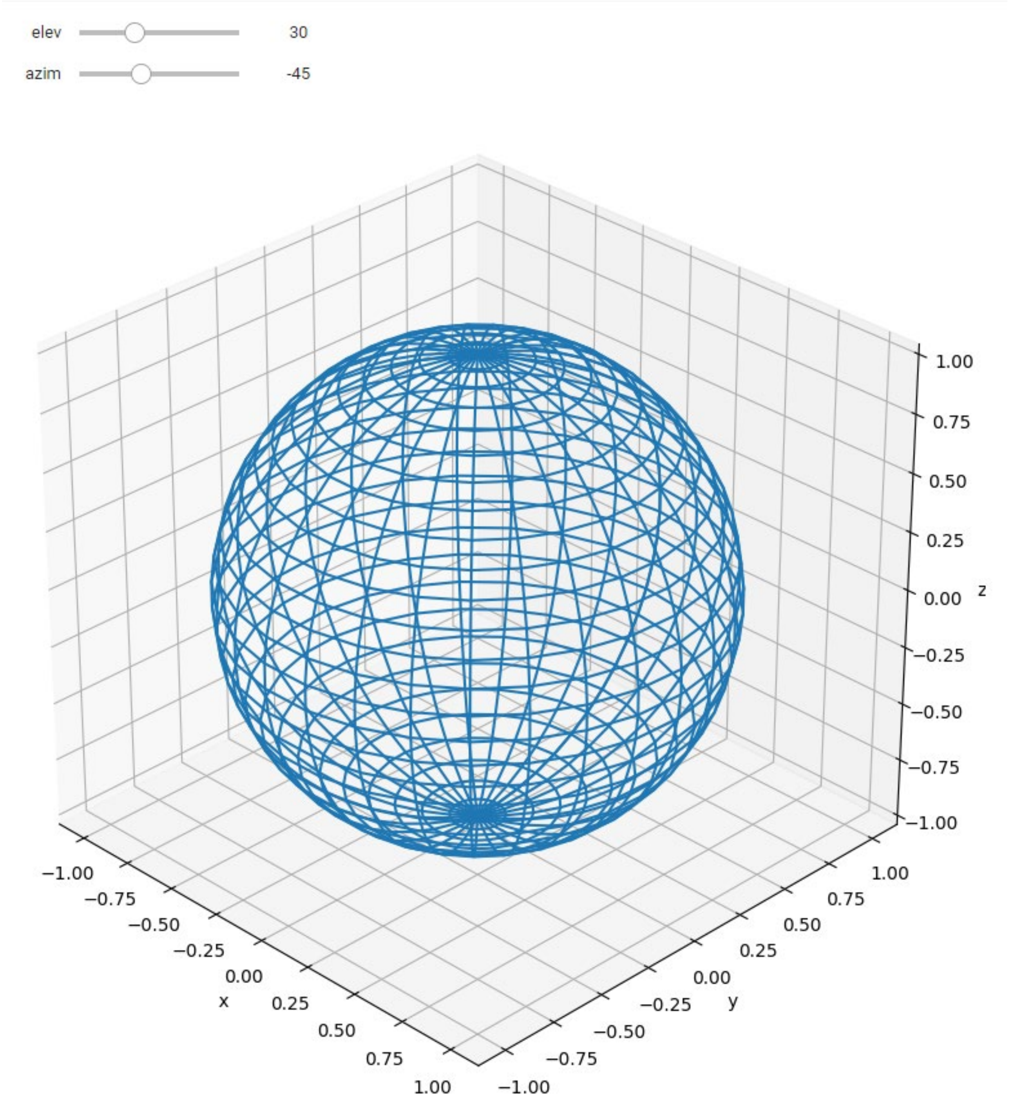


Define a function to draw the 3D wire frame graph using `ipywidgets` interactive sliders

Notice we let `matplotlib` determine which vertices comprise which facets

```
# Cell 5
def plot_wireframe(elev=30, azim=-45):
    ax = plt.axes(projection="3d")
    ax.view_init(elev=elev, azim=azim)
    ax.figure.set_size_inches(10, 10)

    ax.plot_wireframe(x, y, z)
    ax.set_xlabel("x")
    ax.set_ylabel("y")
    ax.set_zlabel("z")
    ax.set_aspect("equal")
    plt.show()


widgets.interactive(plot_wireframe, azim=(-180, 180, 5), elev=(0, 90, 5))
```

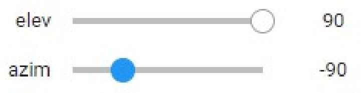
Check plot3d_sphere.ipynb – **Cell 5**



elev 30

azim -45

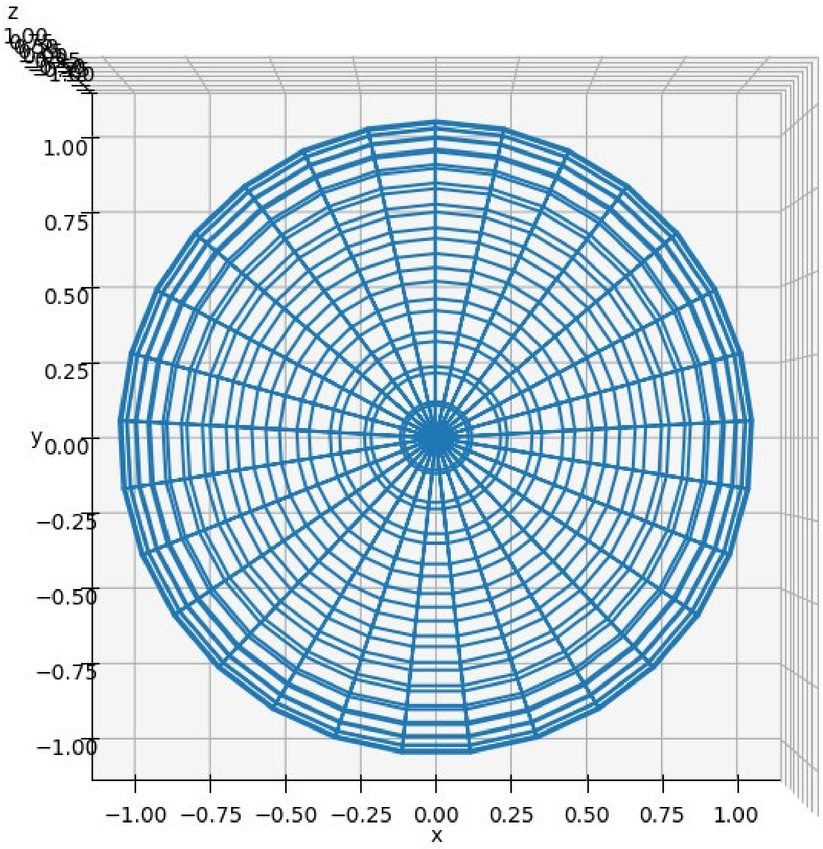
Check plot3d_sphere.ipynb – **Cell 5**



elev 90

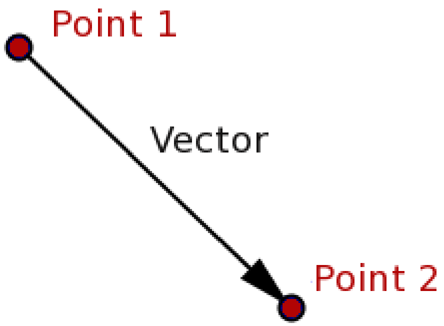
azim -90

①

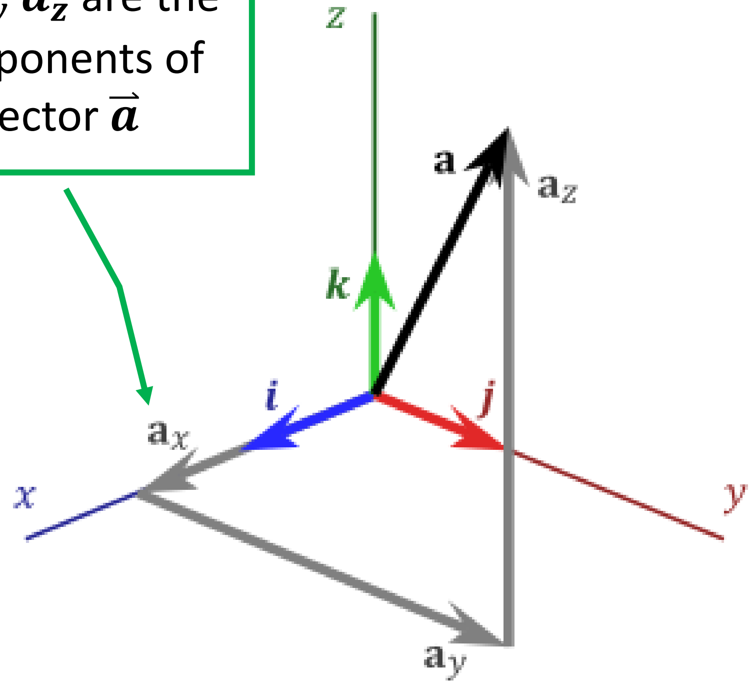


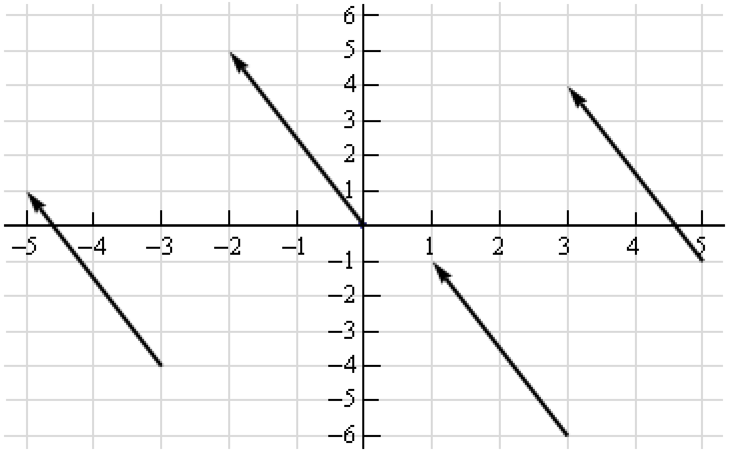
Top-Down View

What is a vector?



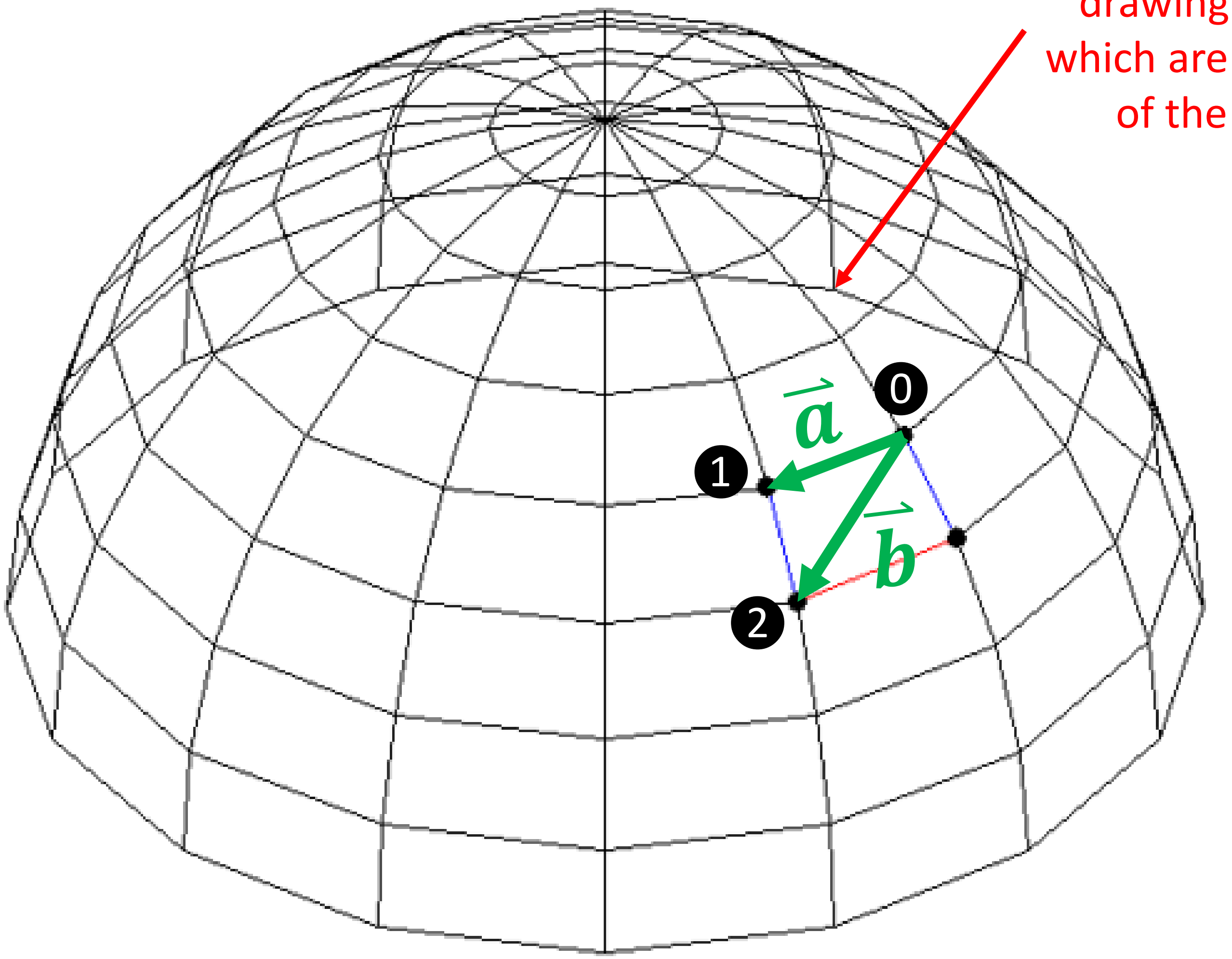
$\boldsymbol{a_x}$ $\boldsymbol{a_y}$ $\boldsymbol{a_z}$ are the components of vector $\vec{\boldsymbol{a}}$





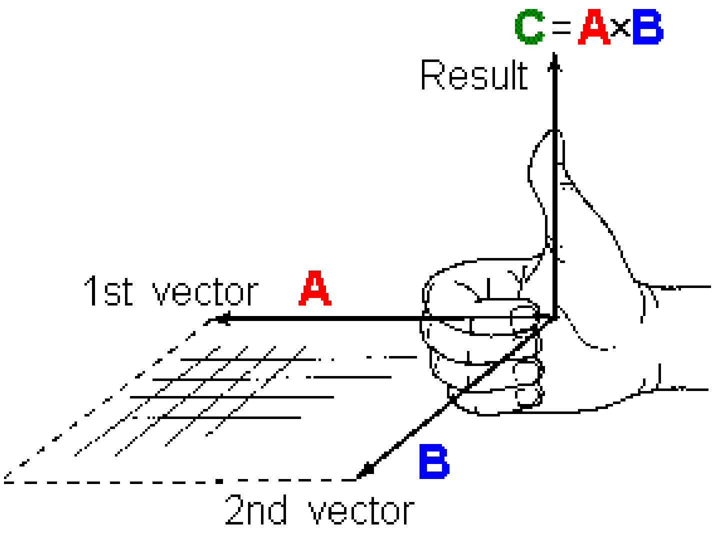
$$\boldsymbol{a_x} = P2_x - P1_x$$
$$\boldsymbol{a_y} = P2_y - P1_y$$
$$\boldsymbol{a_z} = P2_z - P1_z$$

What is a vector?



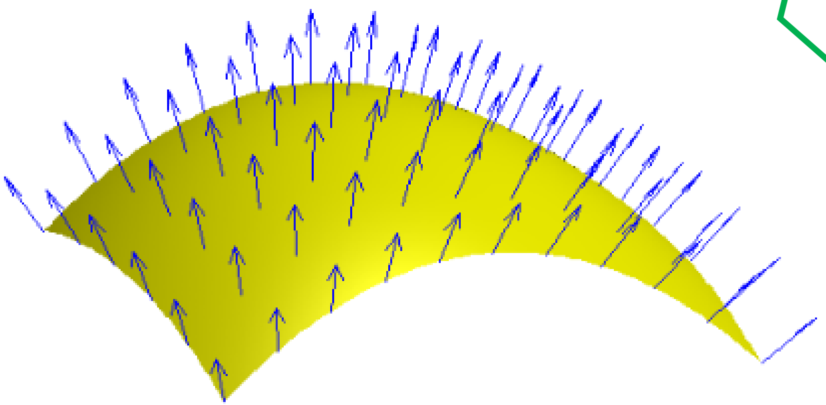
How do we avoid drawing the facets which are on the back of the sphere?

Vector Cross Product



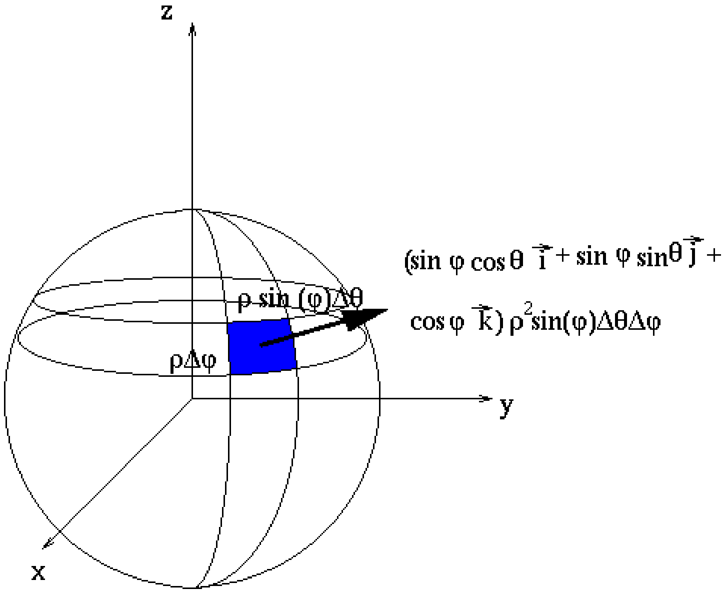
$$c = \begin{vmatrix} i & j & k \\ a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix}$$

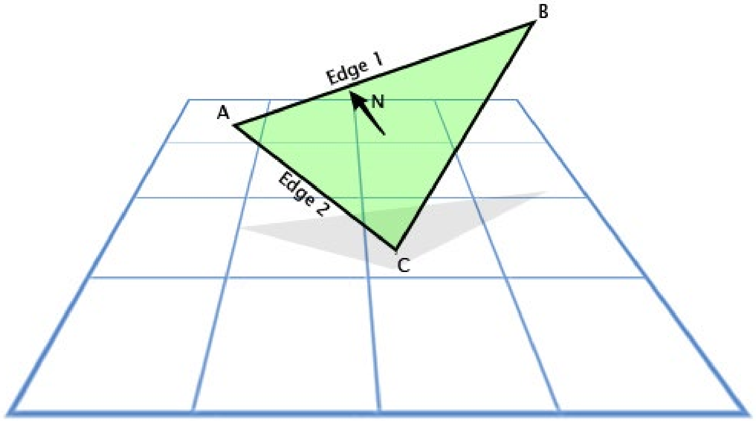
$$c = \begin{matrix} [(a_2 \times b_3) - (a_3 \times b_2)]\, i\ + \\ [(a_3 \times b_1) - (a_1 \times b_3)]\, j\ + \\ [(a_1 \times b_2) - (a_2 \times b_1)]\, k \end{matrix}$$

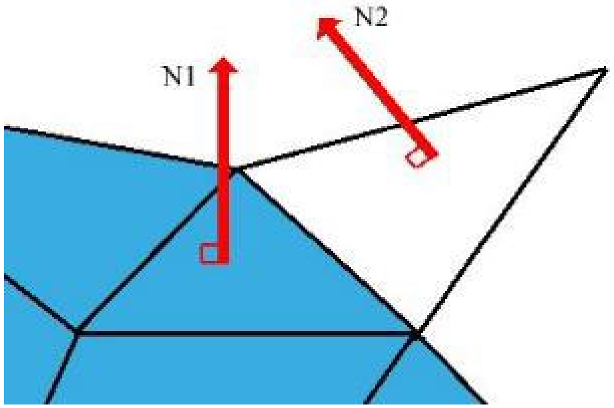


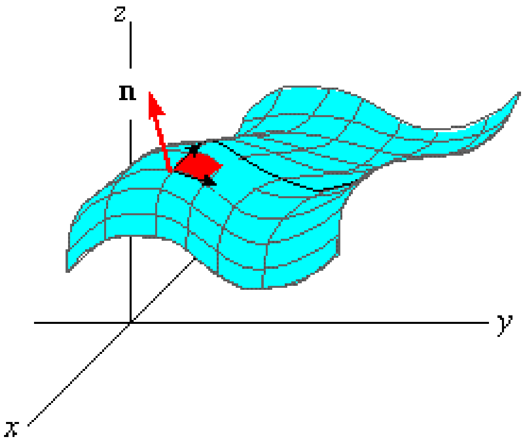
The **cross product** of two vectors is another **vector** which is perpendicular to both vectors **A** and **B**

Every Facet has a Surface Normal Vector

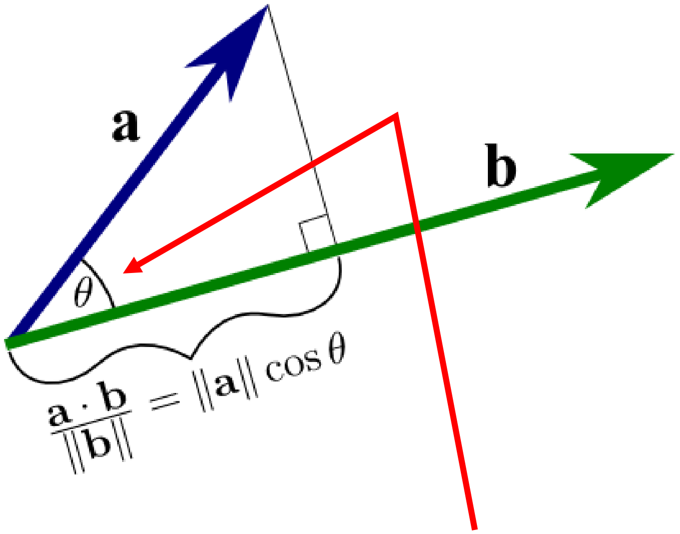

z
(sin φ cos θ $\vec{i}$ + sin φ sinθ $\vec{j}$ +
ρ sin (φ)Δθ
cos φ $\vec{k}$) ρ^{2}sin(φ)ΔθΔφ
ρΔφ
y
x







Vector Dot Product


a
b
θ
$\frac{\mathbf{a}\cdot\mathbf{b}}{\|\mathbf{b}\|} = \|\mathbf{a}\| \cos\theta$

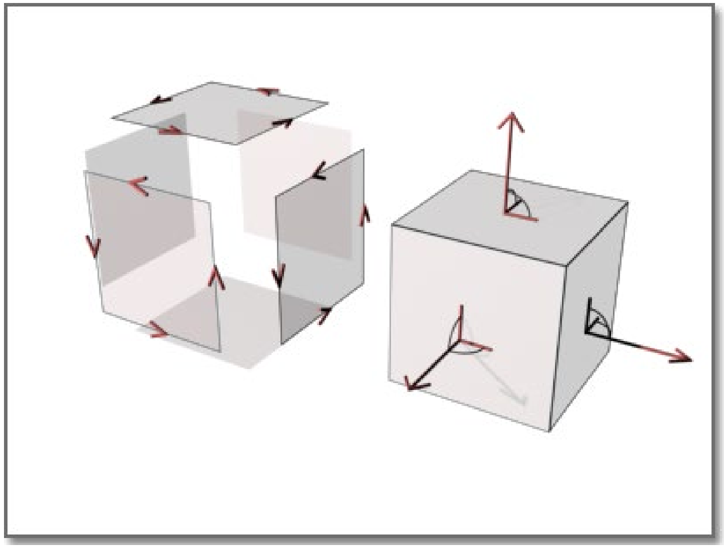
The **dot product** gives the angle *between* two **vectors**

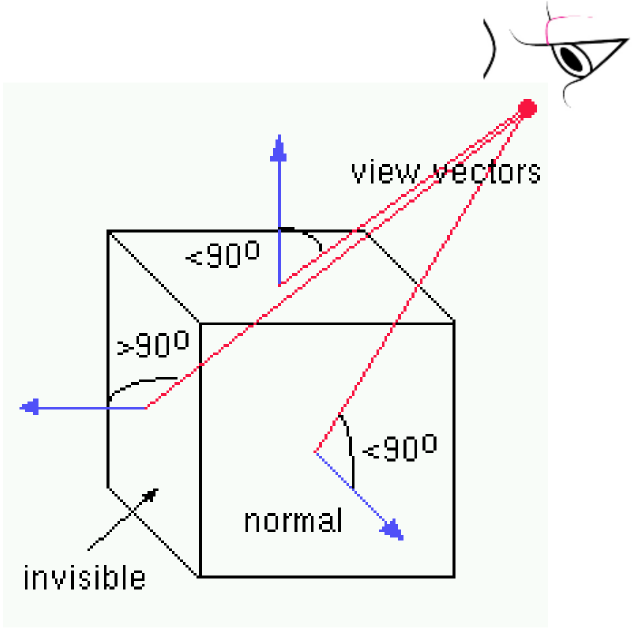
$$\mathbf{A}\cdot\mathbf{B} = \|\mathbf{A}\|\,\|\mathbf{B}\|\cos\theta$$

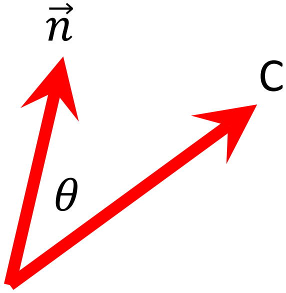
$$\cos\theta = \frac{\mathbf{A}\cdot\mathbf{B}}{\|\mathbf{A}\|\,\|\mathbf{B}\|}$$

$$\theta = \text{acos}\left(\frac{\mathbf{A}\cdot\mathbf{B}}{\|\mathbf{A}\|\,\|\mathbf{B}\|}\right)$$

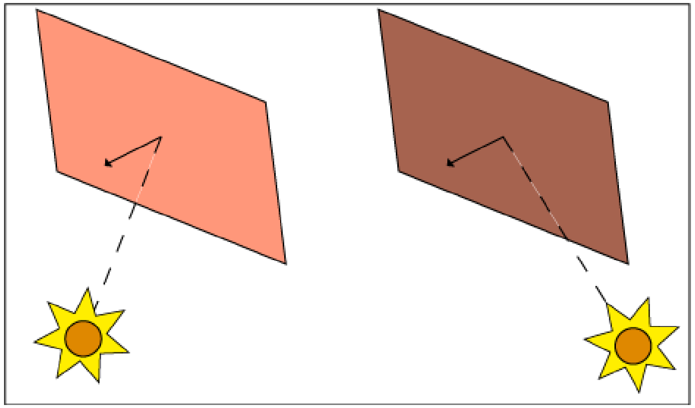
Back Face Culling and Facet Shading



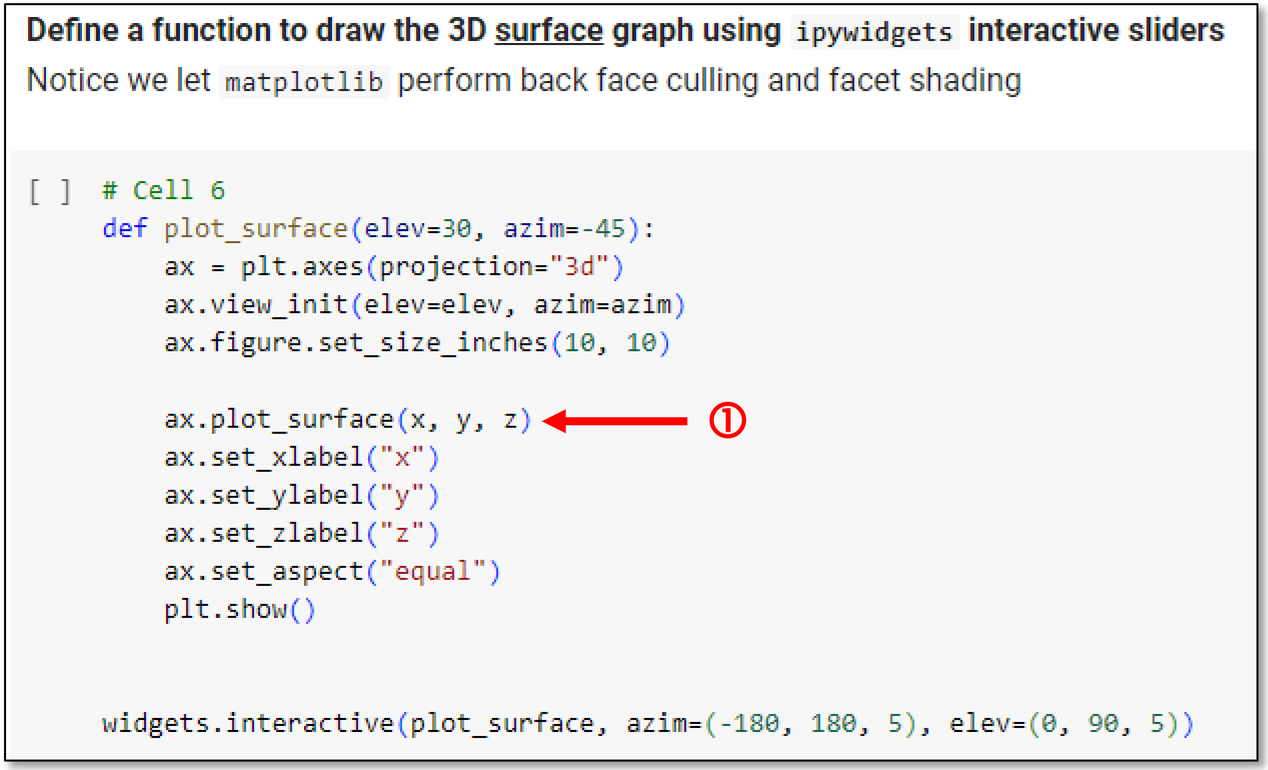




Camera / Eye
Location



Run plot3d_sphere.ipynb – **Cell 6**



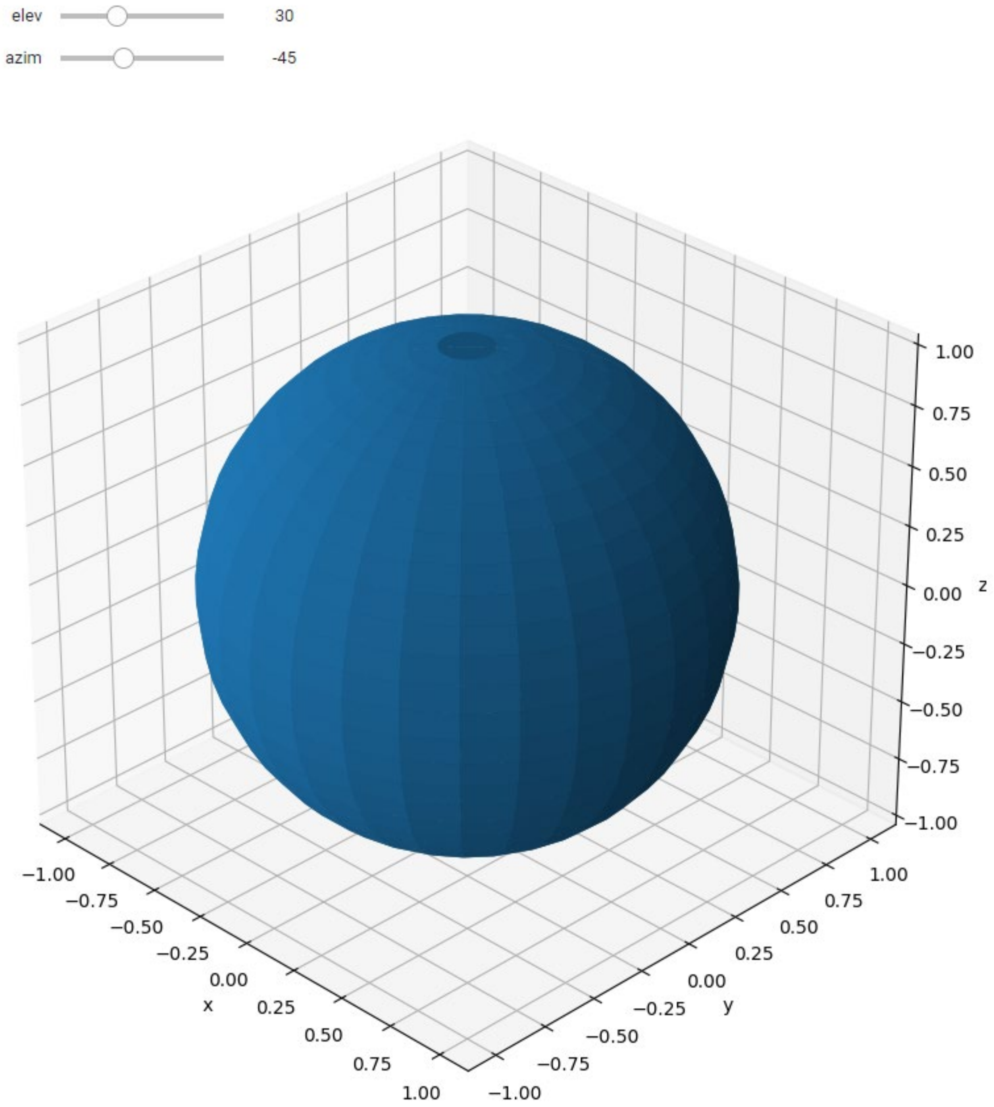
Define a function to draw the 3D surface graph using `ipywidgets` interactive sliders

Notice we let `matplotlib` perform back face culling and facet shading

```
[ ]  # Cell 6
     def plot_surface(elev=30, azim=-45):
         ax = plt.axes(projection="3d")
         ax.view_init(elev=elev, azim=azim)
         ax.figure.set_size_inches(10, 10)

         ax.plot_surface(x, y, z)
         ax.set_xlabel("x")
         ax.set_ylabel("y")
         ax.set_zlabel("z")
         ax.set_aspect("equal")
         plt.show()


     widgets.interactive(plot_surface, azim=(-180, 180, 5), elev=(0, 90, 5))
```

①

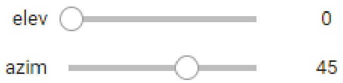
Check plot3d_sphere.ipynb – **Cell 6**



elev 30

azim -45

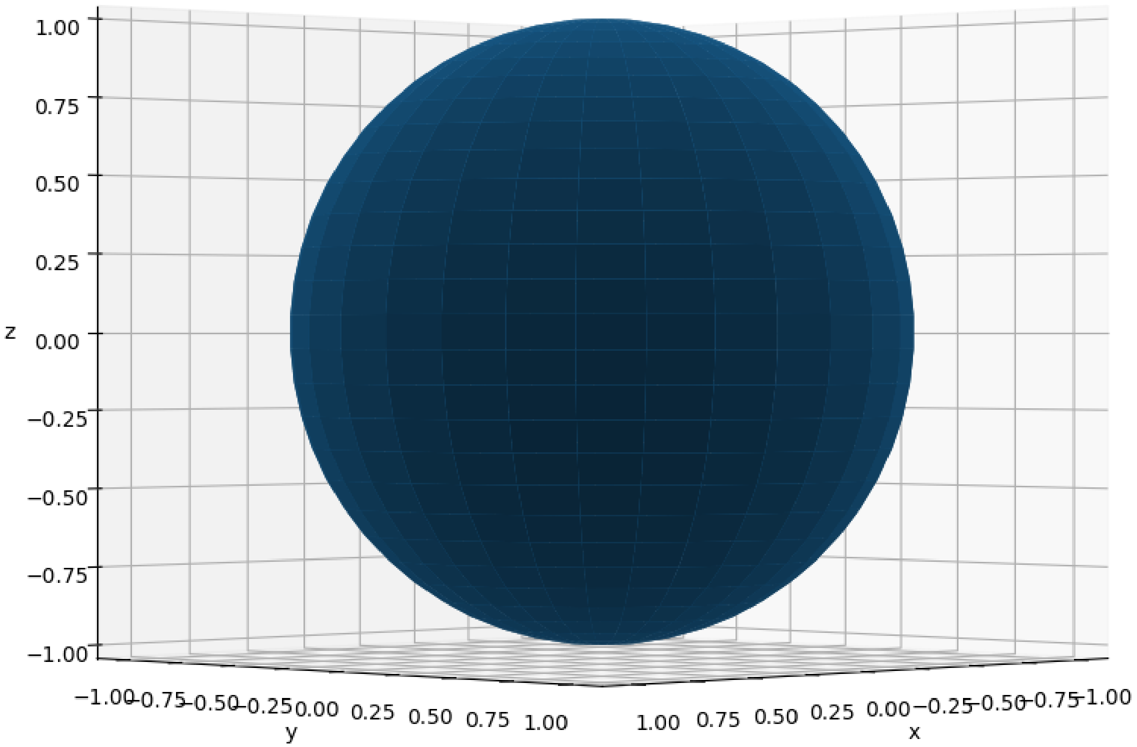
Check plot3d_sphere.ipynb – **Cell 6**



elev 0

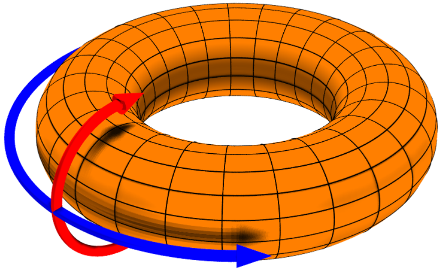
azim 45

①



Modelling a Torus

- Your scientist would like to begin modelling the electromagnetic field around a **toroidal** coil carrying AC current
- The first step will be defining and drawing a 3D torus using a modified version of the spherical coordinate system
- The red arrow points in the **poloidal** direction and the blue arrow points in the **toroidal** direction
- A sphere and a torus are not **homeomorphic**: unlike a sphere, a torus needs <u>two</u> radii to fully describe it

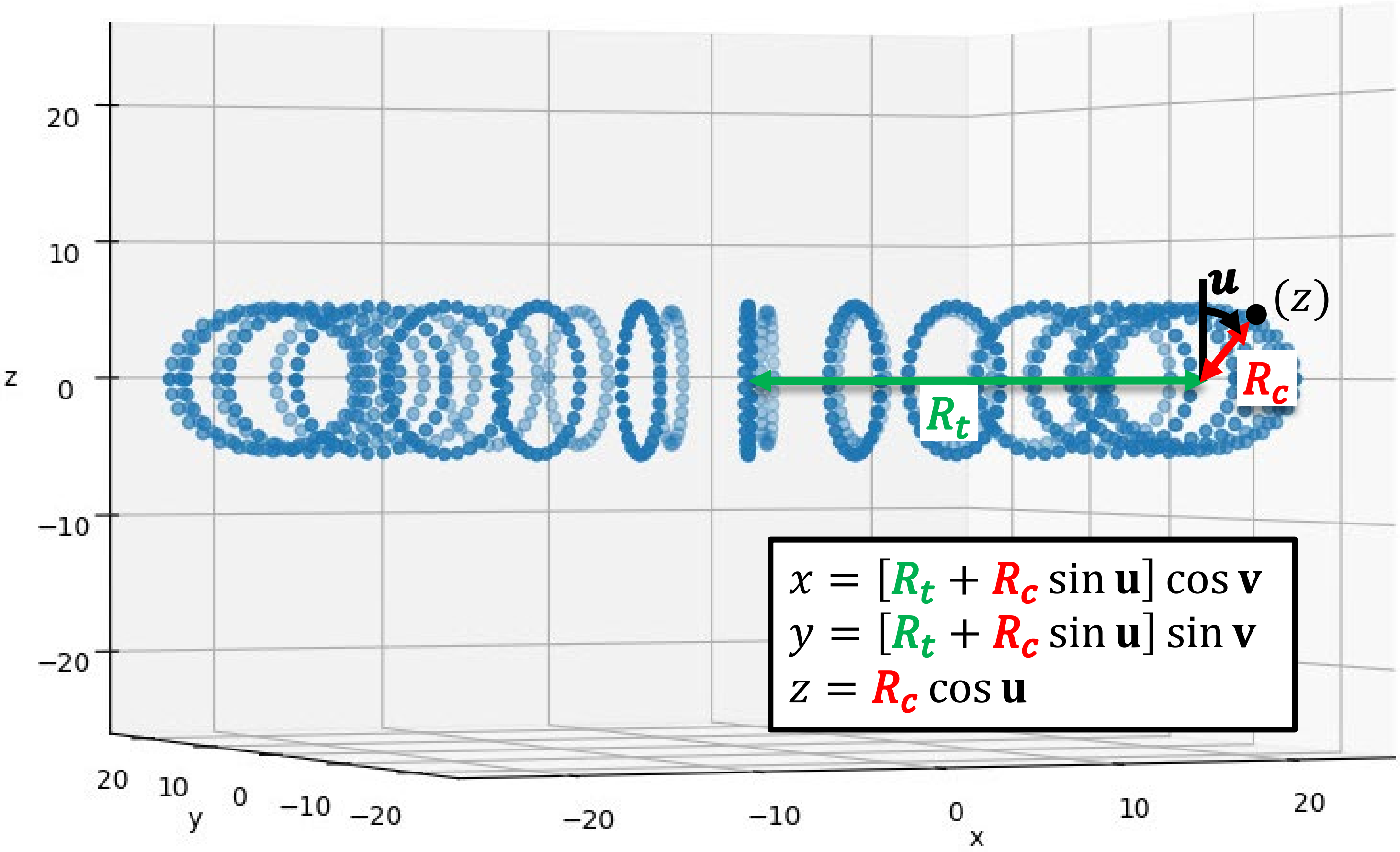


Modelling a Torus

Note: $\mathbf{0 \leq u \leq 2\pi}$

R_t = Radius of Torus

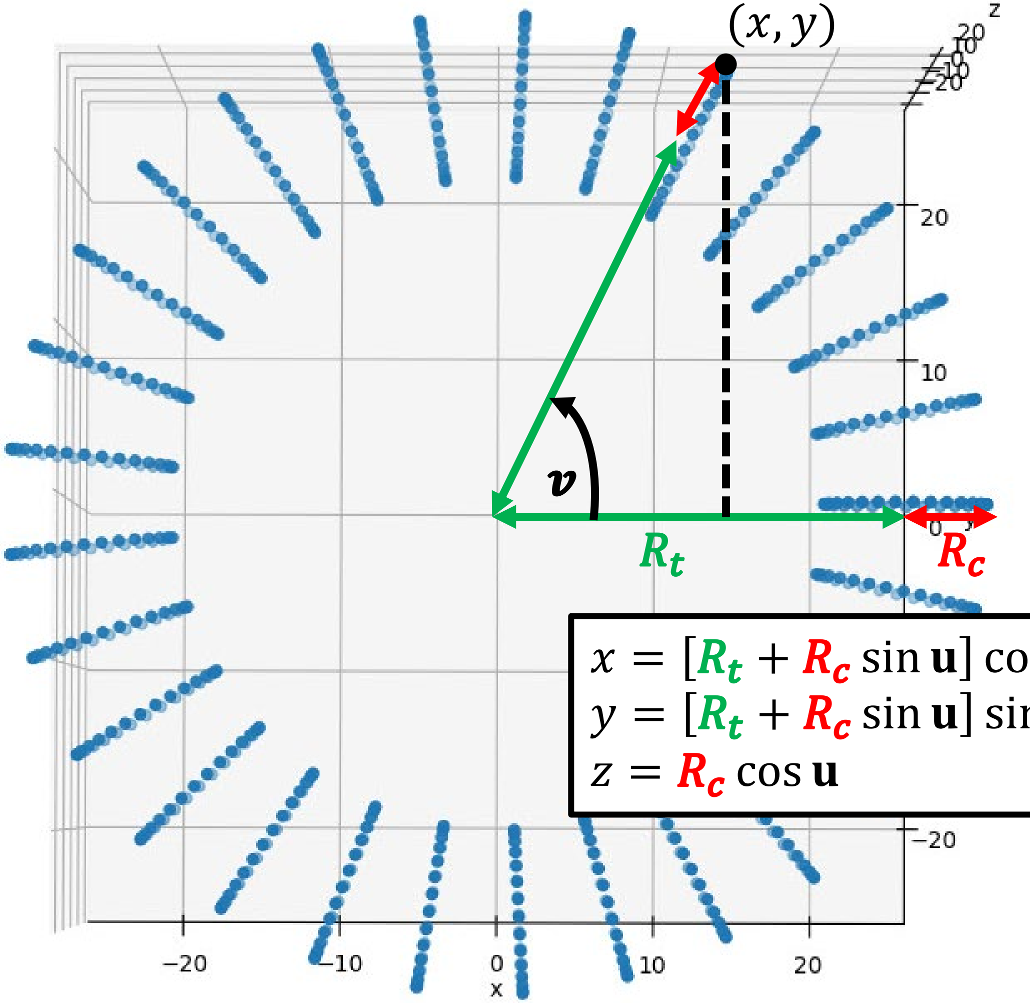
R_c = *Radius of Cross Section*



$$x = [R_t + R_c \sin \mathbf{u}] \cos \mathbf{v}$$
$$y = [R_t + R_c \sin \mathbf{u}] \sin \mathbf{v}$$
$$z = R_c \cos \mathbf{u}$$

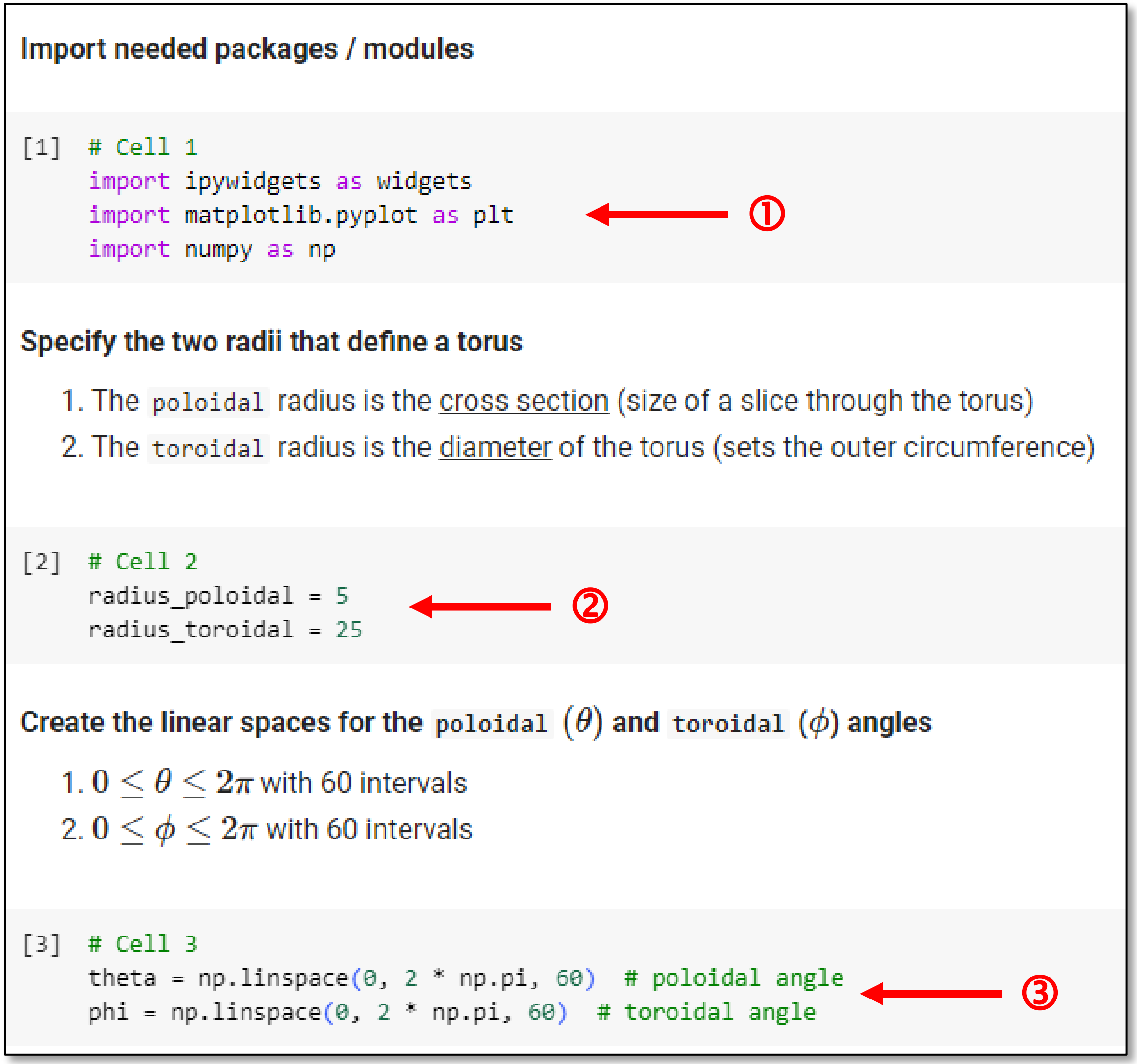
Modelling a Torus

Note: $\mathbf{0 \leq u \leq 2\pi}$



$$x = [R_t + R_c \sin \mathbf{u}] \cos \mathbf{v}$$
$$y = [R_t + R_c \sin \mathbf{u}] \sin \mathbf{v}$$
$$z = R_c \cos \mathbf{u}$$

Run plot3d_torus.ipynb – Cells 1..3



Import needed packages / modules

```
[1]  # Cell 1
     import ipywidgets as widgets
     import matplotlib.pyplot as plt
     import numpy as np
```

①

Specify the two radii that define a torus

1. The `poloidal` radius is the cross section (size of a slice through the torus)
2. The `toroidal` radius is the diameter of the torus (sets the outer circumference)

```
[2]  # Cell 2
     radius_poloidal = 5
     radius_toroidal = 25
```

②

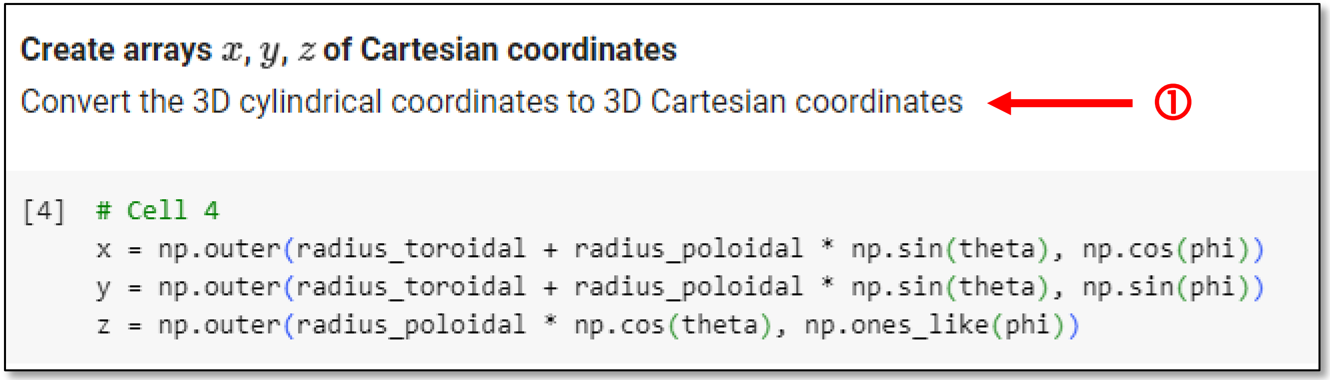
Create the linear spaces for the `poloidal` (θ) and `toroidal` (ϕ) angles

1. $0 \leq \theta \leq 2\pi$ with 60 intervals
2. $0 \leq \phi \leq 2\pi$ with 60 intervals

```
[3]  # Cell 3
     theta = np.linspace(0, 2 * np.pi, 60)  # poloidal angle
     phi = np.linspace(0, 2 * np.pi, 60)  # toroidal angle
```

③

Run plot3d_torus.ipynb – Cell 4



Create arrays x, y, z of Cartesian coordinates

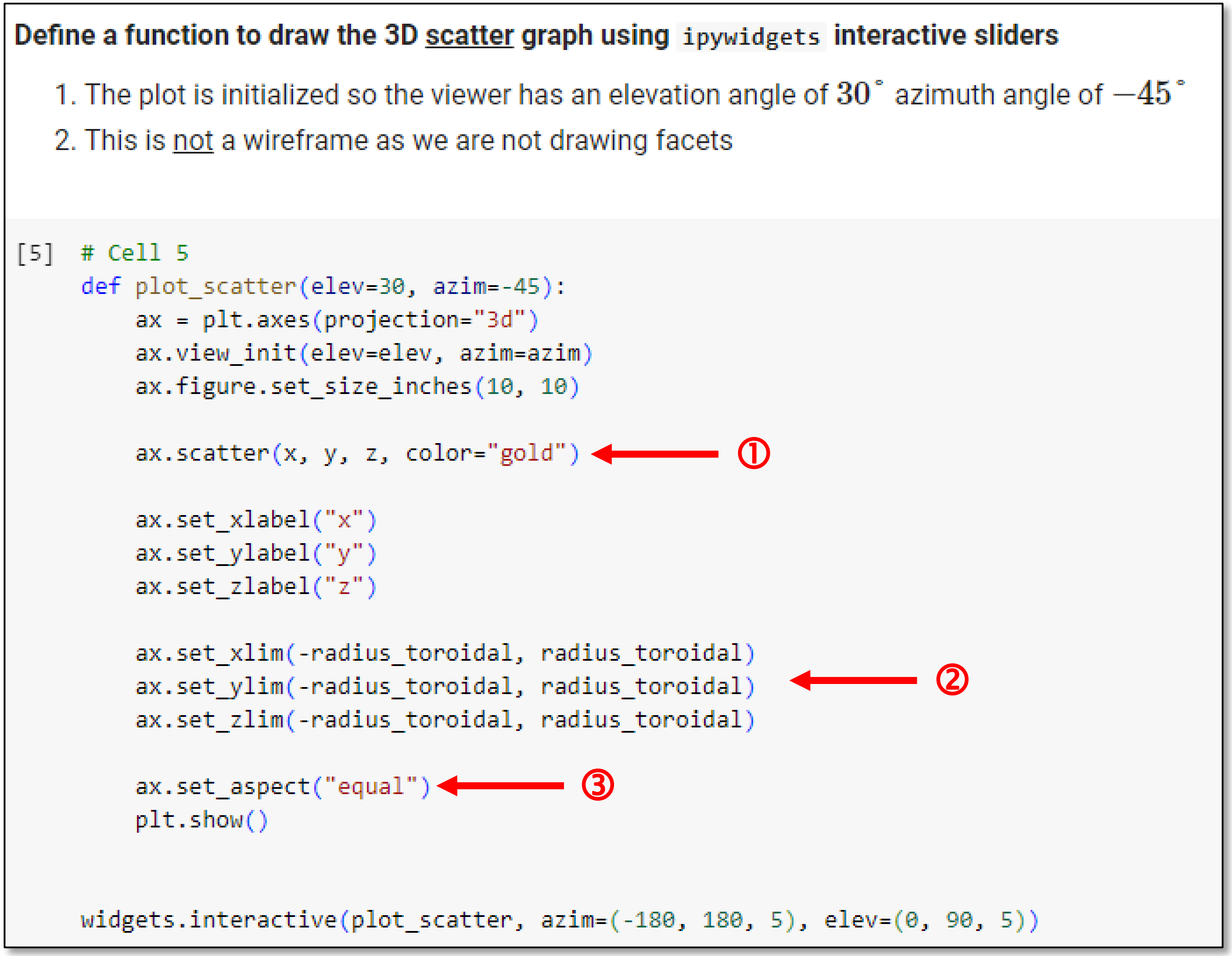
Convert the 3D cylindrical coordinates to 3D Cartesian coordinates ←

①

```
[4]  # Cell 4
     x = np.outer(radius_toroidal + radius_poloidal * np.sin(theta), np.cos(phi))
     y = np.outer(radius_toroidal + radius_poloidal * np.sin(theta), np.sin(phi))
     z = np.outer(radius_poloidal * np.cos(theta), np.ones_like(phi))
```

$$\boxed{\begin{aligned} x &= [\boldsymbol{R_t} + \boldsymbol{R_c} \sin \mathbf{u}] \cos \mathbf{v} \\ y &= [\boldsymbol{R_t} + \boldsymbol{R_c} \sin \mathbf{u}] \sin \mathbf{v} \\ z &= \boldsymbol{R_c} \cos \mathbf{u} \end{aligned}}$$

Run plot3d_torus.ipynb – Cell 5

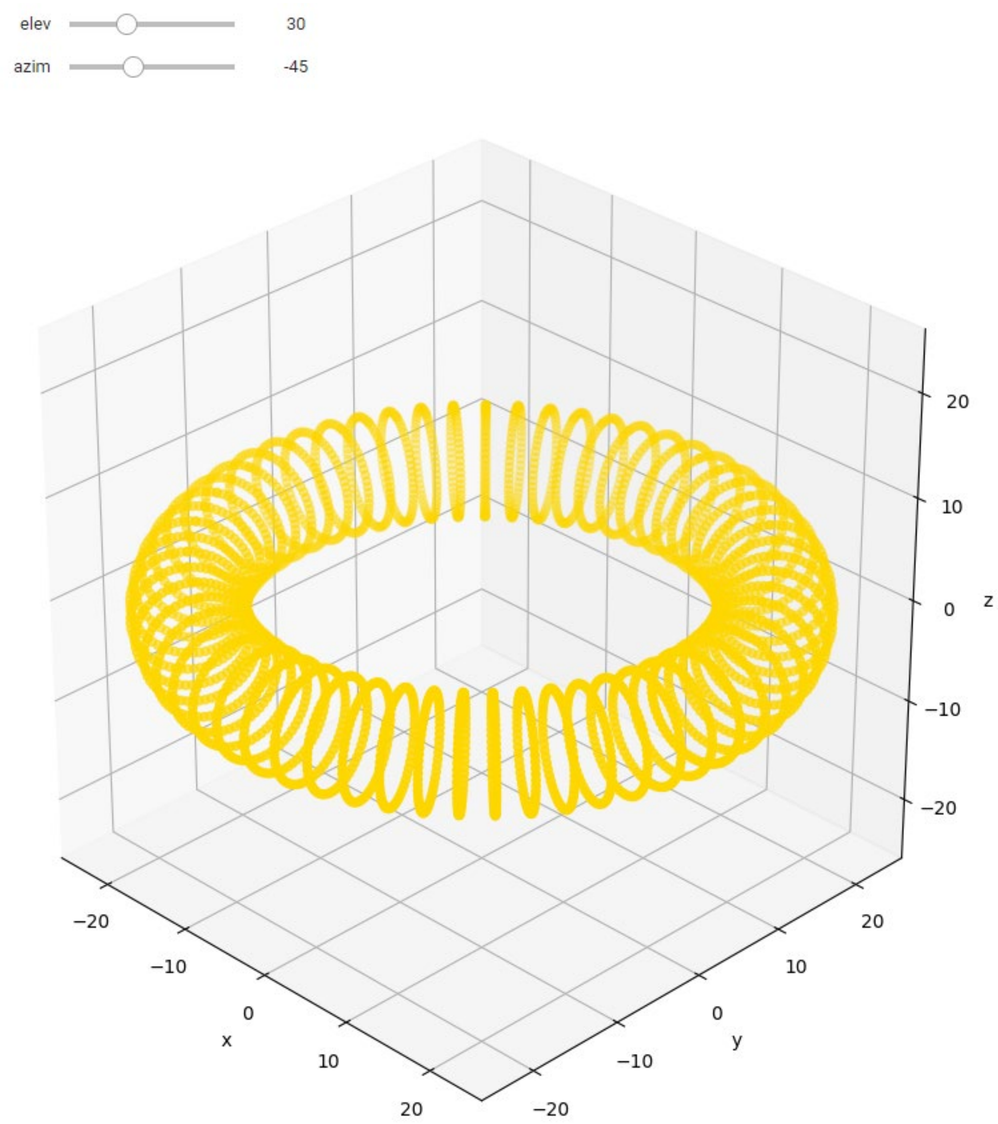


Define a function to draw the 3D scatter graph using `ipywidgets` interactive sliders

1. The plot is initialized so the viewer has an elevation angle of 30° azimuth angle of -45°
2. This is not a wireframe as we are not drawing facets

```
[5]  # Cell 5
     def plot_scatter(elev=30, azim=-45):
         ax = plt.axes(projection="3d")
         ax.view_init(elev=elev, azim=azim)
         ax.figure.set_size_inches(10, 10)

         ax.scatter(x, y, z, color="gold")

         ax.set_xlabel("x")
         ax.set_ylabel("y")
         ax.set_zlabel("z")

         ax.set_xlim(-radius_toroidal, radius_toroidal)
         ax.set_ylim(-radius_toroidal, radius_toroidal)
         ax.set_zlim(-radius_toroidal, radius_toroidal)

         ax.set_aspect("equal")
         plt.show()


     widgets.interactive(plot_scatter, azim=(-180, 180, 5), elev=(0, 90, 5))
```

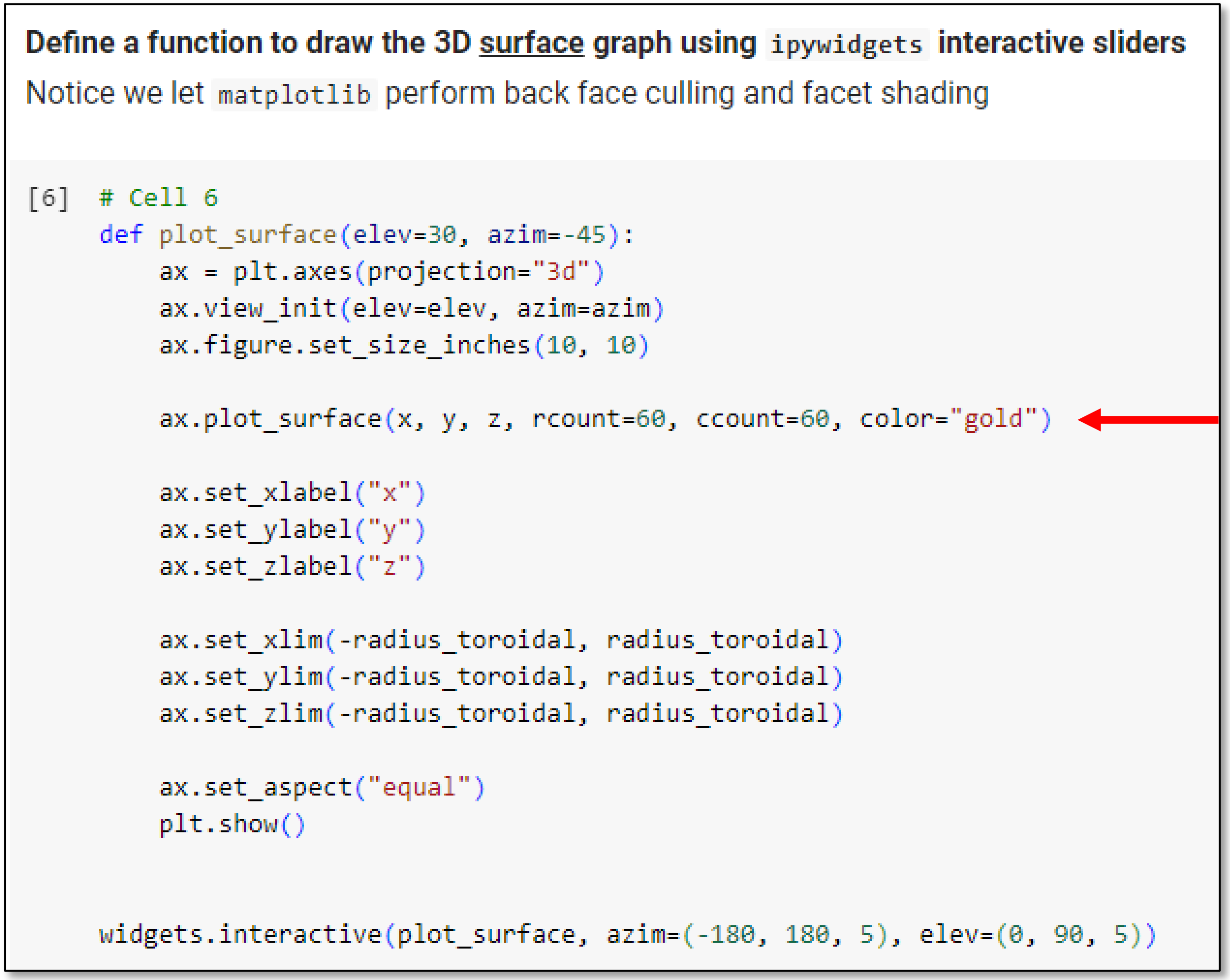
Check plot3d_torus.ipynb – **Cell 5**



elev 30

azim -45

Run plot3d_torus.ipynb – Cell 6



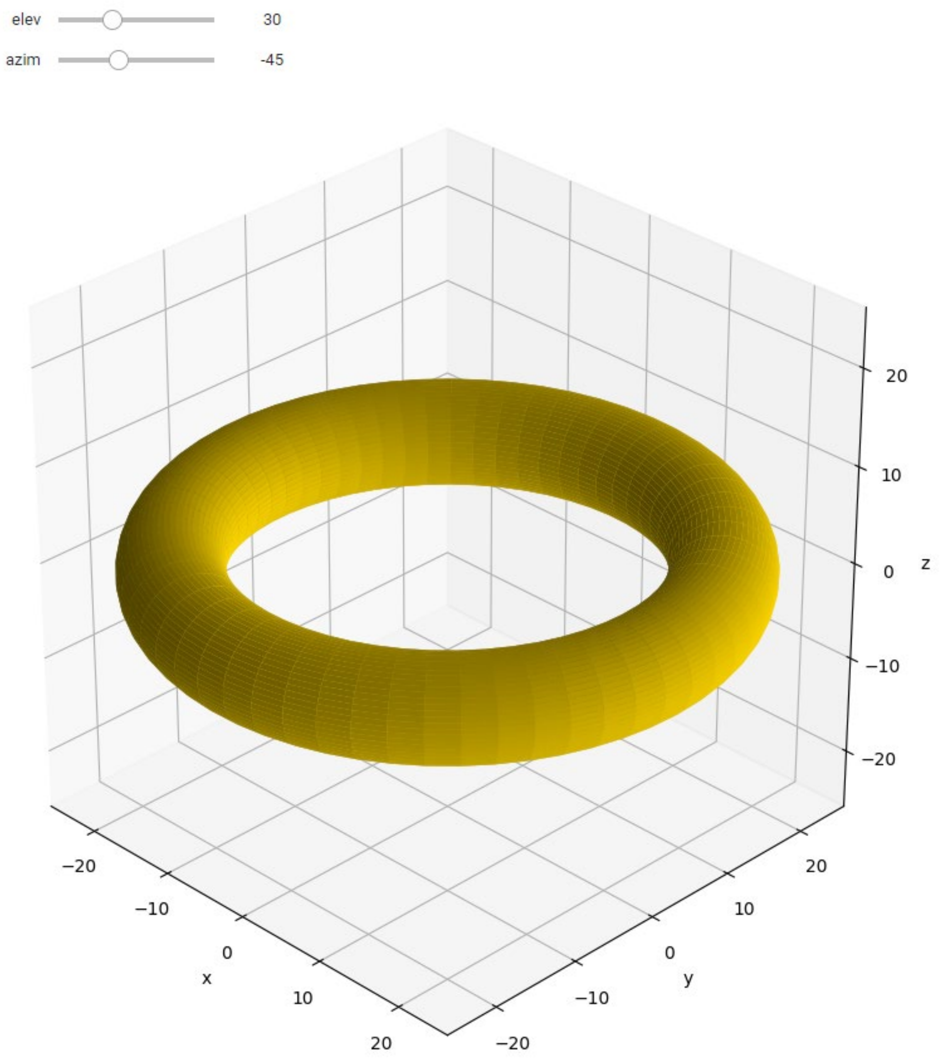
Define a function to draw the 3D surface graph using `ipywidgets` interactive sliders

Notice we let `matplotlib` perform back face culling and facet shading

```
[6]  # Cell 6
     def plot_surface(elev=30, azim=-45):
         ax = plt.axes(projection="3d")
         ax.view_init(elev=elev, azim=azim)
         ax.figure.set_size_inches(10, 10)

         ax.plot_surface(x, y, z, rcount=60, ccount=60, color="gold")

         ax.set_xlabel("x")
         ax.set_ylabel("y")
         ax.set_zlabel("z")

         ax.set_xlim(-radius_toroidal, radius_toroidal)
         ax.set_ylim(-radius_toroidal, radius_toroidal)
         ax.set_zlim(-radius_toroidal, radius_toroidal)

         ax.set_aspect("equal")
         plt.show()


     widgets.interactive(plot_surface, azim=(-180, 180, 5), elev=(0, 90, 5))
```

①

Check plot3d_torus.ipynb – Cell 6



elev 30

azim -45

Session **01** – Now You Know…

- Create numerical **arrays** and plot **polynomials**
- Estimate and plot **infinite series** to visualize **convergence**
- Calculate Euclid’s **GCD** (HCF) of pairs of random integers
- Calculate the 2nd central moment of **uniform distributions**
- Demonstrate **Euler’s Identity** for Complex Numbers
- Use **Polar Coordinates** to draw parametric curves and 2D **random walks**
- Plot the **superposition** of two waves to create *traveling* and standing waves
- Use trigonometry to draw a 3D **sphere** and **torus**