



# **ENGR 21:**

# **Computer Engineering Fundamentals**

Lecture 9  
Tuesday, September 30, 2025

---

# Recap `numpy`



# Conventions for numpy

Three approaches to importing the package

Write at the top of any Python file that will use numpy one of the following

```
import numpy
```

Must use `numpy.<function>`

```
from numpy import *
```

Can simply use `<function>`

```
import numpy as np
```

Must use `np.<function>`

# Features of the numpy package

Regular Python: 'float'

- Data types for numerical computing
  - float16
  - float32
  - float64
  - int8, int16, int32, int64
  - uint8, uint16, uint32, uint64
- Vast library of functions useful for engineering
- Everything is 'array-native' –
  - works seamlessly on large sets of numbers

uint: unsigned integer  
int: integer

```
>>> import numpy

>>> import math

>>> numpy.linspace(1,5,9)

array([1. , 1.5, 2. , 2.5, 3. , 3.5, 4. ,
       4.5, 5. ])

>>> numpy.logspace(1,3,3)

array([ 10., 100., 1000.])

>>> math.sqrt(range(1,10))

TypeError: must be real number, not range

>>> numpy.sqrt(range(1,10))

array([1.          , 1.41421356, 1.73205081,
       2.          , 2.23606798,
       2.44948974, 2.64575131, 2.82842712, 3.
       ])
```

---

# Recap `pip`, Python's native package manager

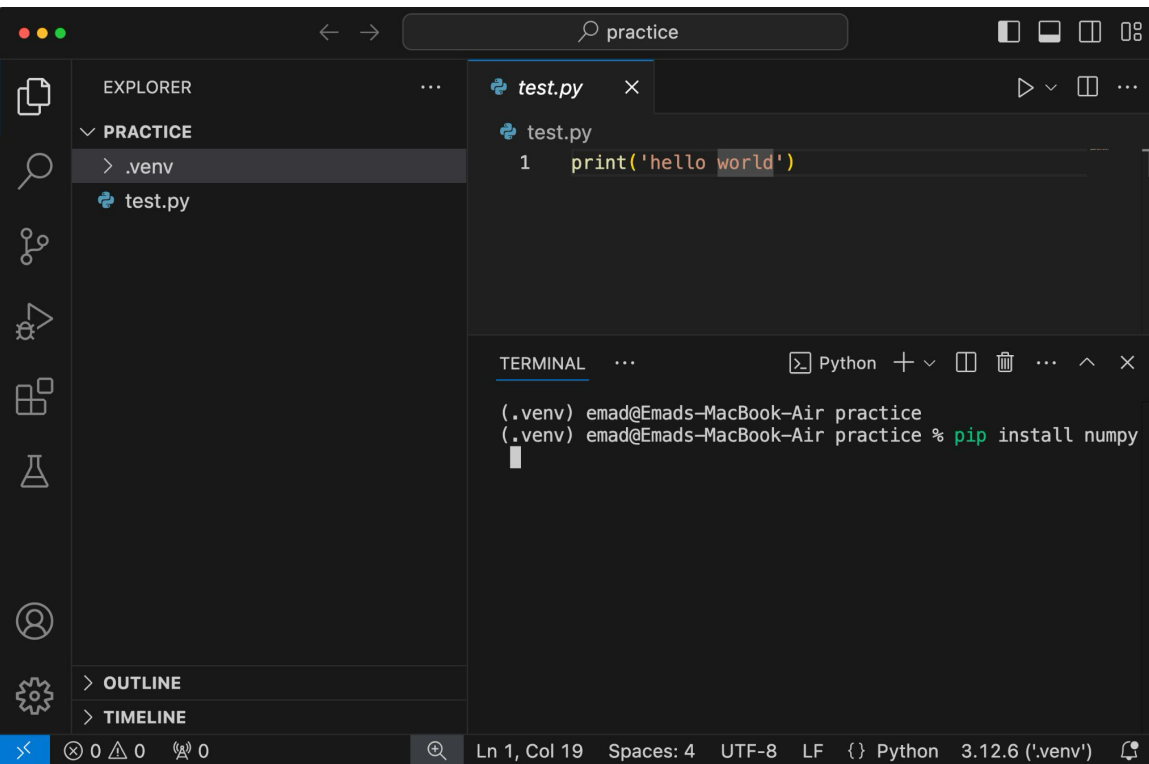
# Installing Packages for Python

Inside a Terminal (NOT Python REPL):

```
pip install <package name>
```

*numpy*  
*matplotlib*

Options for Terminals:



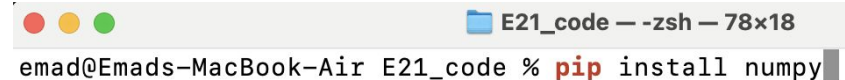
The screenshot shows the Visual Studio Code interface. The Explorer panel on the left shows a project named 'PRACTICE' with a subdirectory '.venv' and a file 'test.py'. The main editor area shows the 'test.py' file with the following code:

```
1 print('hello world')
```

Below the editor is a terminal window titled 'TERMINAL' with a Python icon. It shows the following commands and output:

```
(.venv) emad@Emads-MacBook-Air practice  
(.venv) emad@Emads-MacBook-Air practice % pip install numpy
```

The status bar at the bottom indicates the file is 'test.py' at line 1, column 19, using UTF-8 encoding, LF line endings, and Python 3.12.6 from the '.venv' environment.

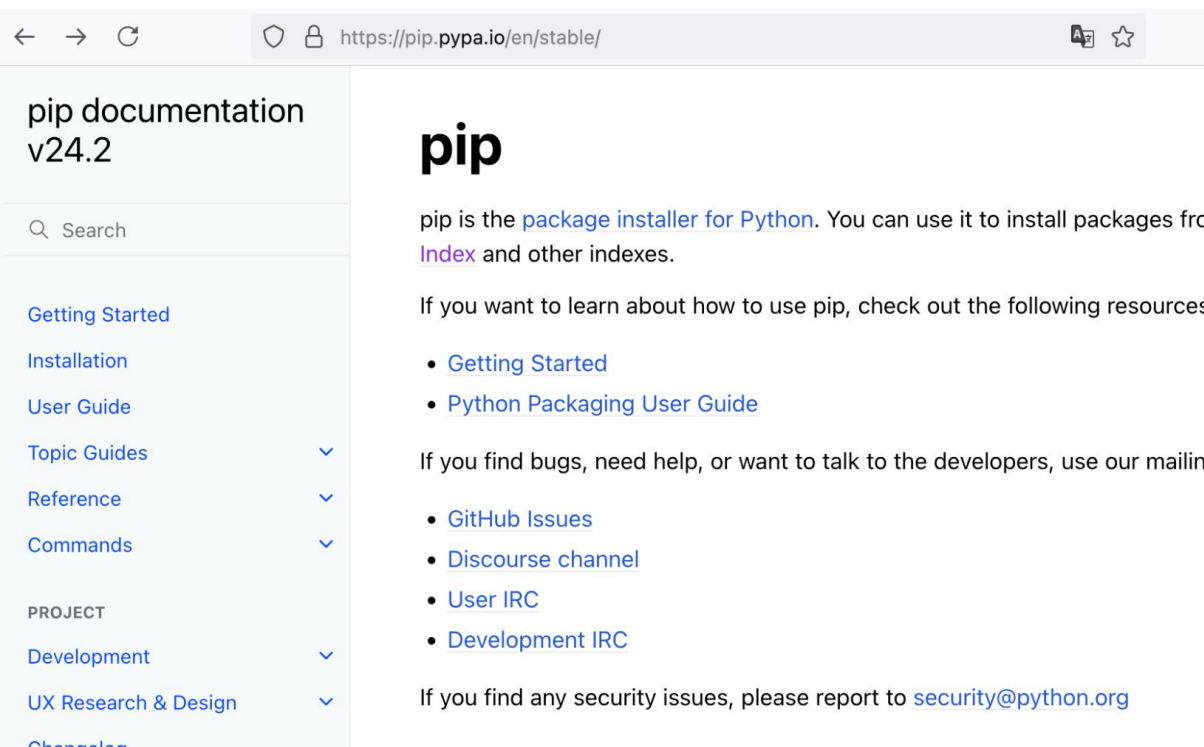


The screenshot shows a terminal window titled 'E21\_code --zsh-- 78x18'. The prompt is 'emad@Emads-MacBook-Air E21\_code %' and the command entered is 'pip install numpy'.

# What does pip do, exactly?

A 'package manager' for Python

Looks up packages on the official **Python Package Index** <https://pypi.org/>



pip documentation v24.2

Search

Getting Started

Installation

User Guide

Topic Guides

Reference

Commands

PROJECT

Development

UX Research & Design

Changelog

## pip

pip is the [package installer for Python](#). You can use it to install packages from [Index](#) and other indexes.

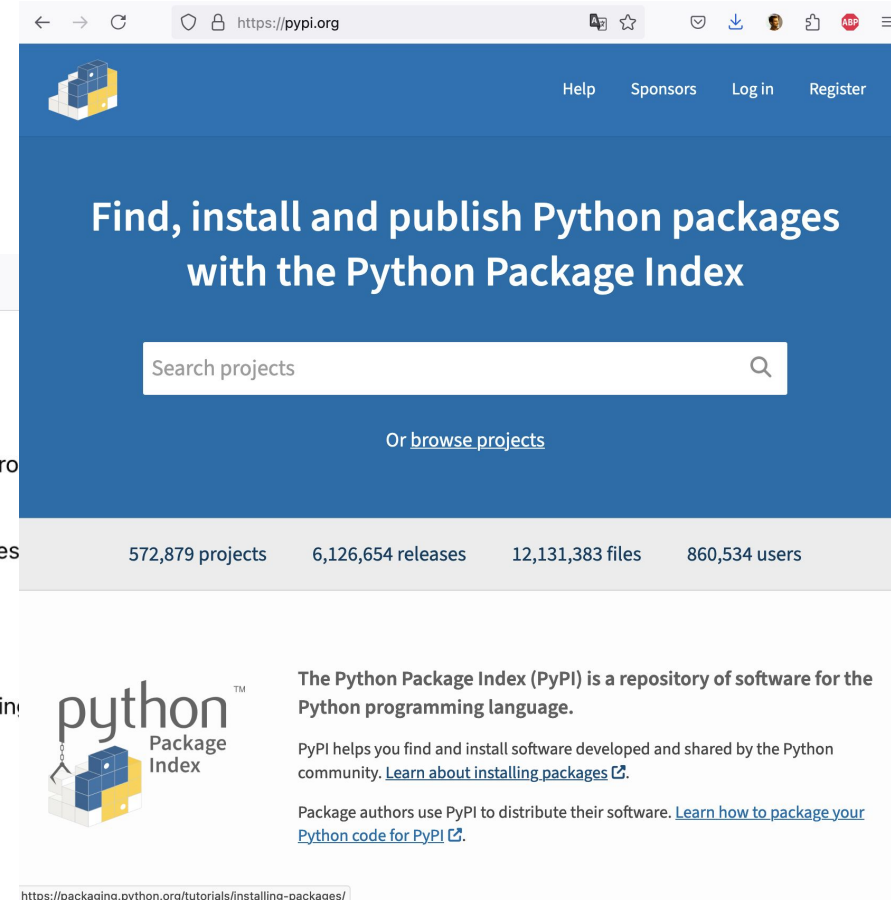
If you want to learn about how to use pip, check out the following resources

- [Getting Started](#)
- [Python Packaging User Guide](#)

If you find bugs, need help, or want to talk to the developers, use our mailing

- [GitHub Issues](#)
- [Discourse channel](#)
- [User IRC](#)
- [Development IRC](#)

If you find any security issues, please report to [security@python.org](mailto:security@python.org)



Help Sponsors Log in Register

## Find, install and publish Python packages with the Python Package Index

Search projects

Or [browse projects](#)

572,879 projects 6,126,654 releases 12,131,383 files 860,534 users

**python**™  
Package Index

The Python Package Index (PyPI) is a repository of software for the Python programming language.

PyPI helps you find and install software developed and shared by the Python community. [Learn about installing packages](#).

Package authors use PyPI to distribute their software. [Learn how to package your Python code for PyPI](#).

<https://packaging.python.org/tutorials/installing-packages/>



## Some useful pip commands

```
pip3 install <package name>  
pip3 list  
pip3 uninstall <package name>
```





# Install numpy on your computer

1. Run `pip3 install numpy` from a terminal
  - a. Either inside VS Code or in your operating system's terminal.
2. Check that numpy works
  - a. Enter Python REPL (enter `python3` into a terminal window)
  - b. Enter `import numpy` at the REPL

---

# Back to `numpy`



# Conventions for numpy

Three approaches to importing the package

Write at the top of any Python file that will use numpy one of the following

```
import numpy
```

Must use `numpy.<function>`

```
from numpy import *
```

Can simply use `<function>`

```
import numpy as np
```

Must use `np.<function>`



Only a convention!  
Can choose any name

# Why does it matter how packages are imported?

Three choices:

```
import numpy
```

```
from numpy import *
```

```
import numpy as np
```

```
>>> import math
>>> import numpy
```

```
>>> sqrt(4)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'sqrt' is not defined
```

```
>>> math.sqrt
<built-in function sqrt>
```

```
>>> numpy.sqrt
<ufunc 'sqrt'>
```

```
>>> math.sqrt(4)
2.0
```

```
>>> numpy.sqrt(4)
np.float64(2.0)
```

# Arrays in numpy

- Like a python list, but better
- Can be multidimensional

■	□	□	■	□	□
□	■	□	□	■	■
■	■	□	■	□	□
□	□	■	□	■	■
■	□	■	□	□	■
□	■	□	■	■	□

## Native Python approach:

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

## Numpy approach

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

Or, we could have made a list of lists ...

# Exploring the 'ndarray' type

zero-indexed

N-dimensional array

list  $\rightarrow$  1-d array

list of lists  $\rightarrow$  2-d array

top level  $\rightarrow$  list  
internal lists

```
>>> a = np.array([[1,2,3],[1.2,3.4,5.6]])  
>>> type(a)  
<class 'numpy.ndarray'>  
>>> numpy.shape(a)  
(2,3)
```

Access elements with '['']'  
 $a[1][1]$   $\rightarrow$  gives '3.4'

row  $\rightarrow$  column

$a[2]$   $\rightarrow$  array (1-d)  
 $a[2][1]$   $\rightarrow$  2<sup>nd</sup> element of this array (numbers.)

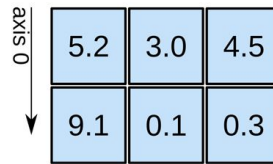
1D array



axis 0

shape: (4,)

2D array

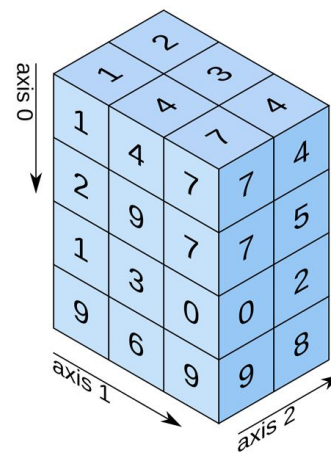


axis 0

axis 1

shape: (2, 3)

3D array



axis 0

axis 1

axis 2

shape: (4, 3, 2)

## Resources

- Resources
  - External Guides and Tutorials
  - Instructor's Circuit Playground Guide for E21
  - Links and Code Snippets
    - Lec 1.1, Tue Sep 2
    - Lec 2.1, Tue Sep 9
    - Lec 2.2, Thu Sep 11
    - Lec 3.1, Tue Sep 16
    - Lec 3.2, Thu Sep 18
    - Lec 4.1, Tue Sep 23
    - Lec 4.2, Thu Sep 25

# Common NumPy functions

Write correct NumPy expressions using each of these functions

`numpy.array()`  
`numpy.zeros()`  
`numpy.ones()`  
`numpy.empty()`  
`numpy.arange()`  
`numpy.linspace()` + *logspace*  
`numpy.random.rand()`  
`numpy.random.randint()`  
`numpy.reshape()`  
`numpy.transpose()`  
`numpy.concatenate()`  
`numpy.flatten()`  
`numpy.resize()`  
`numpy.shape()`  
`numpy.savetxt()`  
`numpy.loadtxt()`

## Resources

- Resources
  - External Guides and Tutorials
  - Instructor's Circuit Playground Guide for E21
  - Links and Code Snippets
    - Lec 1.1, Tue Sep 2
    - Lec 2.1, Tue Sep 9
    - Lec 2.2, Thu Sep 11
    - Lec 3.1, Tue Sep 16
    - Lec 3.2, Thu Sep 18
    - Lec 4.1, Tue Sep 23
    - Lec 4.2, Thu Sep 25

# Floating Point Error Accumulation

in 16-bit floats. Higher-Precision floats have the same problem.

Floating-point numbers have some inherent limitations.

10,000 times

$$0.001 + \dots + 0.001 = 4.0 ?$$

$$10000 * 0.001 = 10.0 ?$$

- Try changing  $n$ .

→ write the IEEE format  
16-bit string of 0's and 1's.

```
import numpy as np

# Use NumPy's float16 type
a = np.float16(1e-3) # A small number
n = 10000 # Large number of iterations

result = np.float16(0) # Start with 0 in float16

# Add the small number 'a' to 'result' n times
for i in range(n):
    result += a

print("Expected result:", a * n) # Correct result with full precision
print("Result with float16:", result) # Result with float16
```

Handwritten annotations on the code:

- A red circle with the number 3 is next to the variable `n`.
- A red box around `1e-3` has an arrow pointing to `4e-3` written in a red box.
- A red bracket next to the loop `for i in range(n):` has the text `a + a + ... + a, n times` written in red.
- Below the loop, `n * a` is written in red.
- A red box around `a * n` in the `print` statement has an arrow pointing to the `4e-3` box.



## Resources

- Resources
  - External Guides and Tutorials
  - Instructor's Circuit Playground Guide for E21
  - Links and Code Snippets
    - Lec 1.1, Tue Sep 2
    - Lec 2.1, Tue Sep 9
    - Lec 2.2, Thu Sep 11
    - Lec 3.1, Tue Sep 16
    - Lec 3.2, Thu Sep 18
    - Lec 4.1, Tue Sep 23
    - Lec 4.2, Thu Sep 25

# Breaking floating-point numbers

Floating-point numbers have some inherent limitations.

$$16 + 0.2 + 0.2 + 0.2 = 128.8 \quad | 16.6 |$$

$$128 + 0.2 + 0.2 + 0.2 = \cancel{128.6} \quad | 128.8$$

try on 128 and 0.2

```
def showFloat(number):
    if str(type(number)) == "<class 'numpy.float16'>":
        x = np.float16(number)
        bits = x.view(np.uint16)
        return f"{bits:016b}"
```

```
import numpy as np

# Declare some 16-bit floating-point numbers
a = np.float16(128)
c = np.float16(16)
ep = np.float16(0.2)

# Add them together and print them out:

print('128 + 0.2 =', a+ep)
print('128 + 0.2 + 0.2 =', a+ep+ep)
print('128 + 0.2 + 0.2 + 0.2 =', a+ep+ep+ep)
print('128 + (0.2 + 0.2 + 0.2) =', a+(ep+ep+ep))
# View the internals
print(showFloat(a+(ep+ep+ep)))
print(showFloat(a+ep+ep+ep))
print('')
print('16 + 0.2 =', c+ep)
print('16 + 0.2 + 0.2 =', c+ep+ep)
print('16 + 0.2 + 0.2 + 0.2 =', c+ep+ep+ep)
print('16 + (0.2 + 0.2 + 0.2) =', c+(ep+ep+ep))
```

128.5

Q why is this better?

Q: Why does problem show up for 128 earlier than for 16?