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`

uint: unsigned integer
int: integer

- Vast library of functions useful for engineering
- Everything is 'array-native' –
  - works seamlessly on large sets of numbers

```
>>> 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

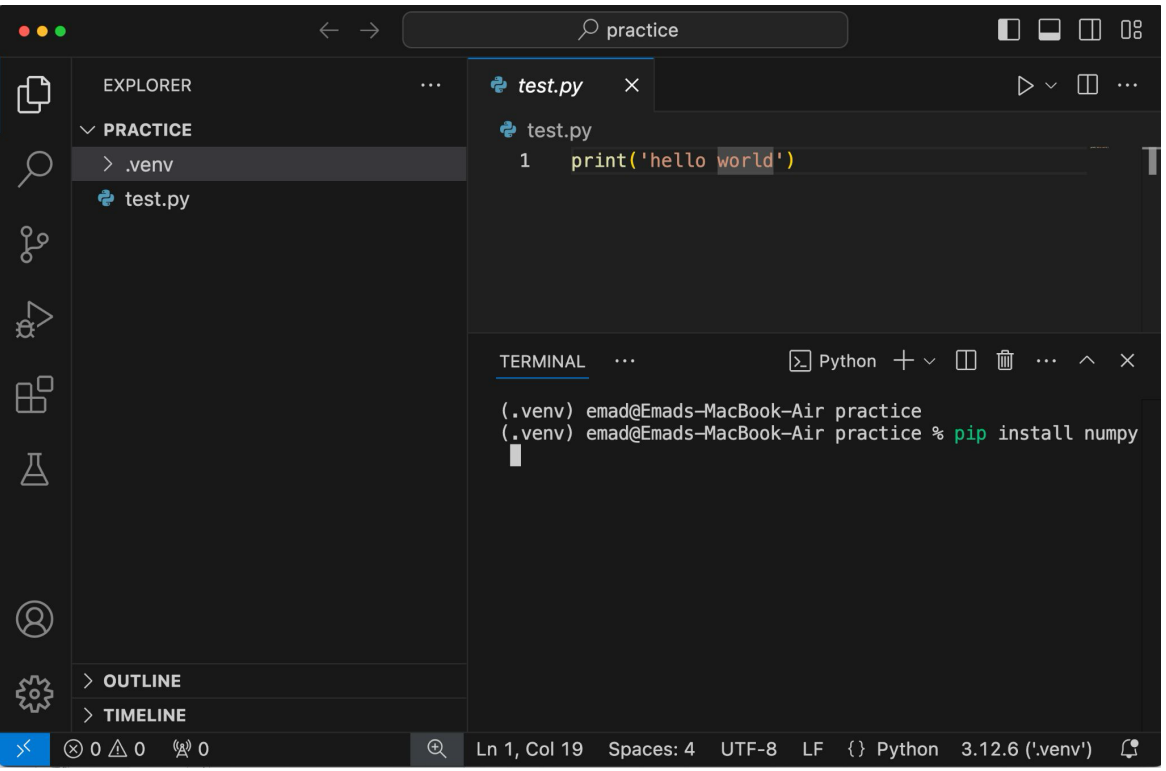
Installing Packages for Python

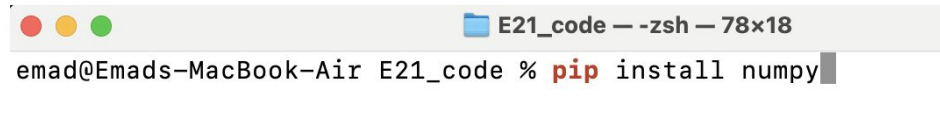
Inside a Terminal (NOT Python REPL):

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

numpy
matplotlib

Options for Terminals:

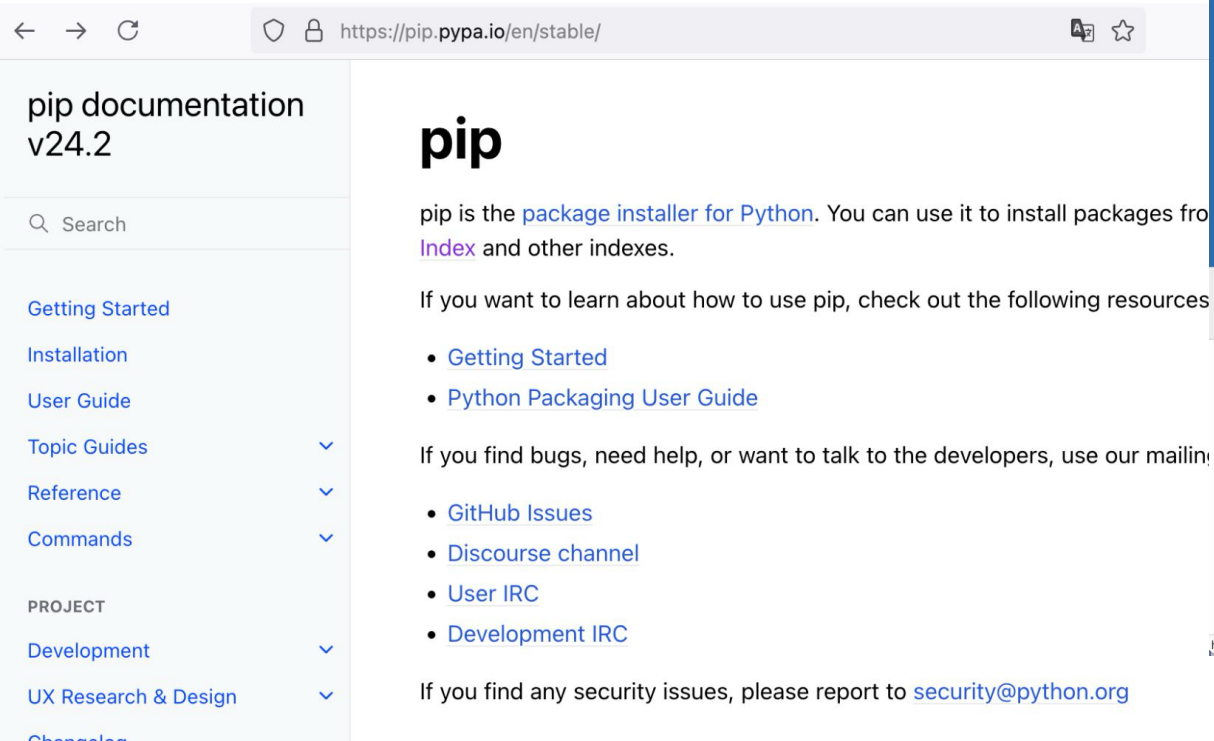


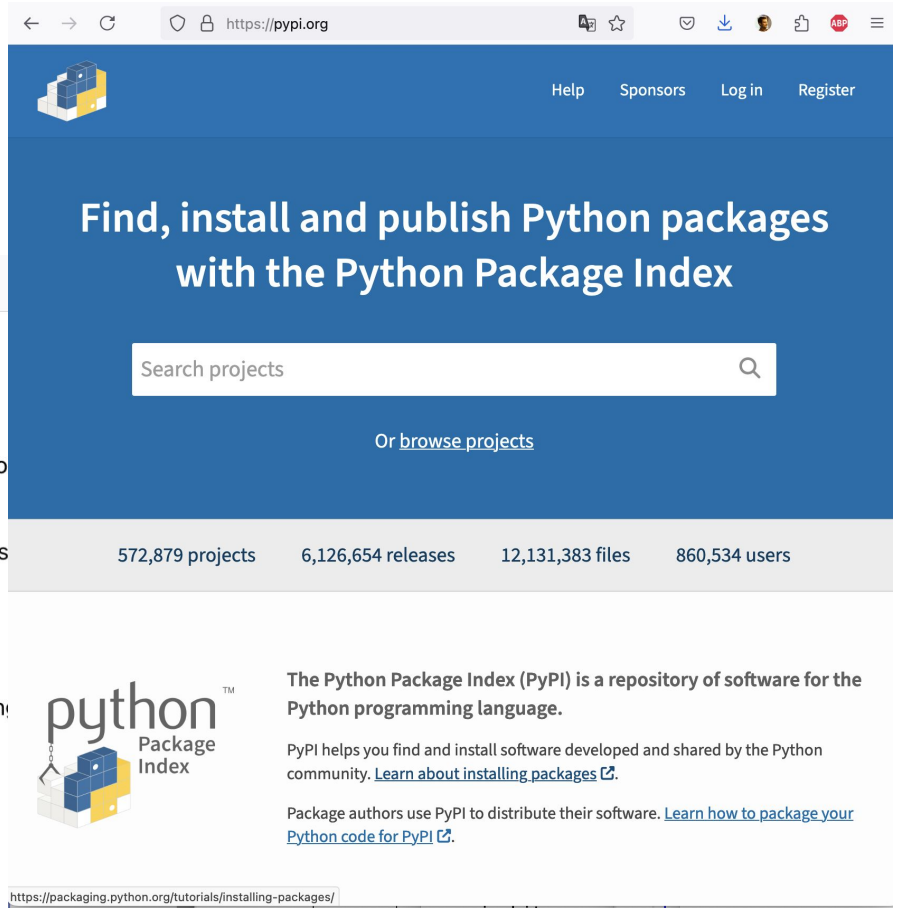


What does `pip` do, exactly?

A ‘package manager’ for Python

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





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
```

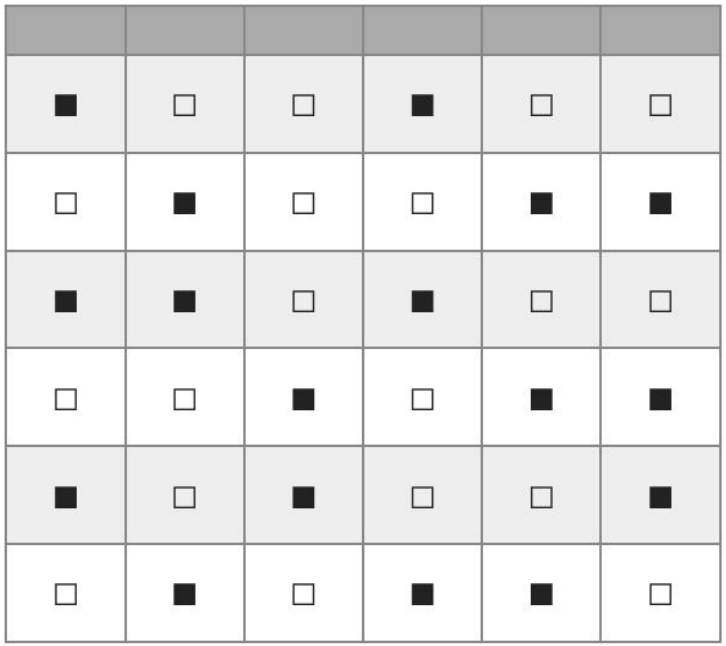
```
from numpy import *
```

```
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 → 1-d array

list of lists → 2-d array

top level 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 '[ ]'

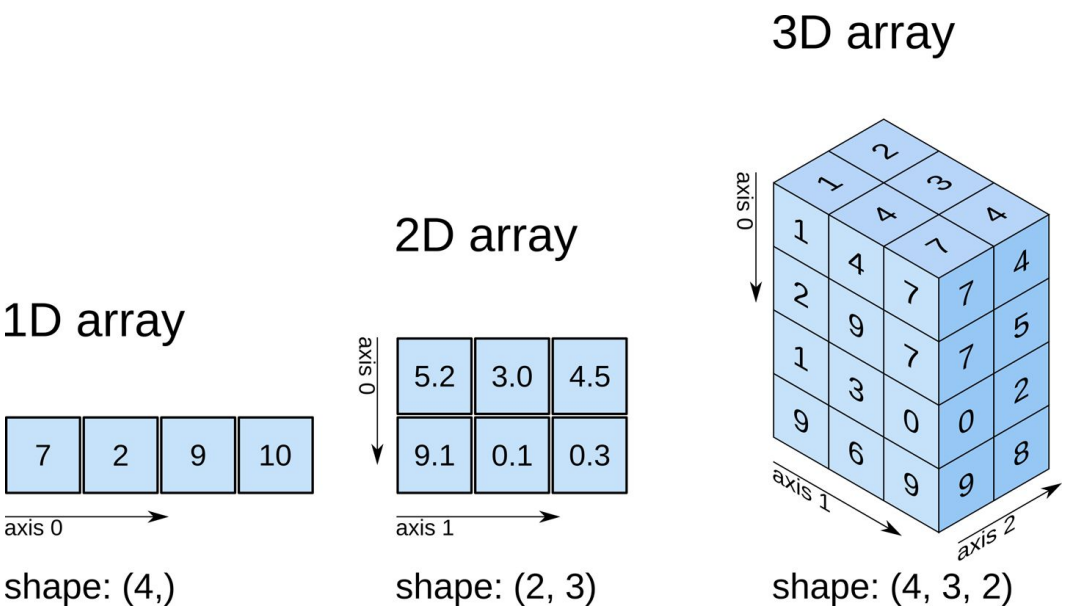
a[1][1] → gives '3.4'

row

column

a[2] → array (1-d)

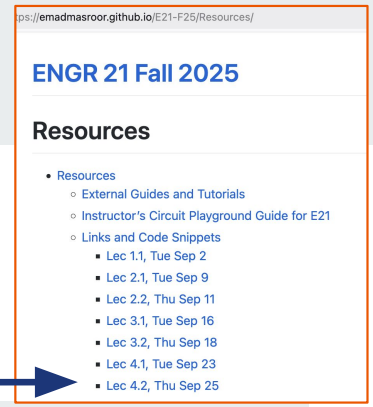
a[2][1] → 2nd element of this array (numbers)



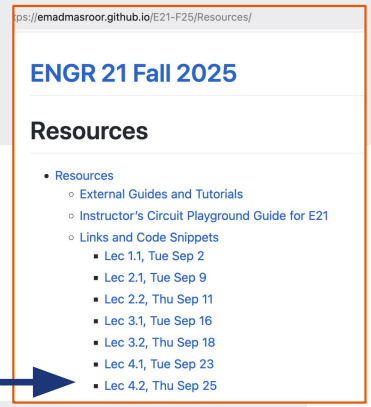
https://www.oreilly.com/library/view/elegant-scipy/9781491922927/ch01.html

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()
```



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 + … + 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.

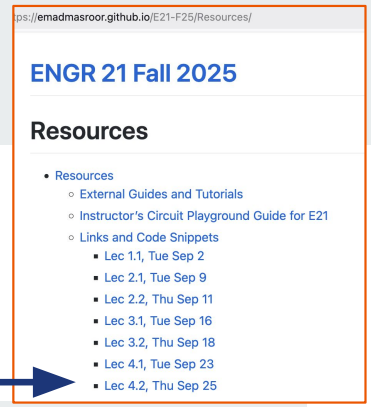
```python
import numpy as np

# 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
```

4e-3

3

a+a+…+a, n times

n x a



Breaking floating-point numbers

Floating-point numbers have some inherent limitations.

```
16 + 0.2 + 0.2 + 0.2 = 128.8
```
16.61

```
128 + 0.2 + 0.2 + 0.2  =
```
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?