Coursework

- **exam2 this week, 2/28-3/2**
  - Study HPL Qs 2, 6.1-10
- **hw05 will be assigned next week**
numpy Redux
Which of the following produces a valid plot given data in x and y?

A) import matplotlib.pyplot as plt
   plt.plot(x, y)
   plt.show()

B) import matplotlib.pyplot as plt
   plt.plot(x, y)
   plt.show()

C) import matplotlib.pyplot as plt
   plt.plot(x, y)
   plt.show()
Which of the following produces a valid plot given data in `x` and `y`?

A  `import matplotlib.pyplot as plt
    plt.plot( x,y )
    plt.show()`  

B  `import matplotlib.pyplot as plt
    plt.plot( x,y )
    plt.show()`  

C  `import matplotlib.pyplot as plt
    plt.plot( x,y )
    plt.show()`
import numpy as np
x = np.array([ 5,1,3 ])
x *= 2

What is the value of x?
A [ 10,2,6 ]
B array( [ 10,2,6 ] )
C [ 5,1,3,5,1,3 ]
D array( [[ 5,1,3 ], [ 5,1,3 ]] )
import numpy as np
x = np.array([5, 1, 3])
x *= 2

What is the value of x?
A [ 10, 2, 6 ]
B ⋆
    array( [ 10, 2, 6 ] )
C [ 5, 1, 3, 5, 1, 3 ]
D array( [[ 5, 1, 3 ], [ 5, 1, 3 ]] )
```python
import numpy as np
x = np.array([ 1 ] * 2)
x += 1
```

What is the final value of `x`?

A. `array([2])`
B. `array([1,1,1])`
C. `array([2,2])`
D. `array([3])`
import numpy as np
x = np.array([1] * 2)
x += 1

What is the final value of x?
A array([2])
B array([1, 1, 1])
C array([2, 2])
D array([3])
Indexing arrays

4 columns

3 rows

5  4  9  2

3  4  1  2

3  2  1  8

- Reminder: numpy indexes by array[row][col].
What will produce this array?

A \texttt{np.array([[1,2,3],[1,2,3]])}

B \texttt{np.array([2,3])}

C \texttt{np.array([3,2])}

D \texttt{np.array([[[1,1],[2,2]],[[3,3]]])}
What will produce this array?

A `np.array([[1,2,3],[1,2,3]])`
B `np.array([2,3])`
C `np.array([3,2])`
D `np.array([[1,1],[2,2],[3,3]])`
numpy supports many possible data types:

- **bool**
- **int16, int32, int64**
- **float16, float32, float64**
- **complex64, complex128**

For the most part, stick with **bool**, **int64**, and **float64** (most accurate).

Specify (and query) with **dtype**:

```python
a = [ 3,2,4 ]
x = np.array( a,dtype=np.float64 )
x.dtype
```
x = np.zeros([2,3])  # zeroes
y = np.ones([4,1])   # ones

- Produce arrays of zeros or ones with specified dimensions.
\[ z = \text{np.eye}(5) \quad \# \text{ identity} \]

- Produces identity matrix of specified square dimension.
\[ w = \text{np.linspace}(0,10,101) \]
\[ v = \text{np.linspace}(\text{start}, \text{finish}, n) \]

- Produce arrays from \text{start} to \text{finish} of \text{n} points (\text{not} spacing!).
- Excellent for grids and coordinates.
- May also see \text{arange}, but I recommend avoiding its use:

\[ u = \text{np.arange}(0,10,0.1) \] # tricky!
\[ u == \text{array}( [ 0, 0.1, 0.2, \ldots, 9.9 ] ) \]
Plot $\sin(x)$ for $x \in [0, 2\pi]$ using pure Python.

```python
from math import pi
x = []  # can't use range easily!
for i in range(100):
    x.append(2*pi*i/100)
from math import sin
y = []
for j in range(100):
    y.append(sin(x[j]))
plt.plot(x,y,'k-')
plt.xlim(0,2*pi)
plt.ylim(-1,1)
plt.show()
```
Looping over 2D arrays

```python
x = np.zeros((3, 3))
for i in range(3):
    for j in range(3):
        x[i][j] = j + 1
```
<table>
<thead>
<tr>
<th></th>
<th>5</th>
<th>2</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>3</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

$m=3$

$n=4$
for i in range(m):
    for j in range(n):
        x[i][j] = 0

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>3</td>
<td>8</td>
</tr>
</tbody>
</table>
for i in range(m):
    for j in range(n):
        x[i][j]=0
for i in range(m):
    for j in range(n):
        x[i][j]=0
for i in range(m):
    for j in range(n):
        x[i][j] = 0
for i in range(m):
    for j in range(n):
        x[i][j]=0

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>2</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>3</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>
for i in range(m):
    for j in range(n):
        x[i][j]=0
for i in range(m):
    for j in range(n):
        x[i][j] = 0
for i in range(m):
    for j in range(n):
        x[i][j] = 0
for i in range(m):
    for j in range(n):
        x[i][j] = 0
for i in range(m):
    for j in range(n):
        x[i][j] = 0

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>3</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>
for i in range(m):
    for j in range(n):
        x[i][j]=0

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>3</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

```
  i   j
  1   0
```
for i in range(m):
    for j in range(n):
        x[i][j] = 0
for i in range(m):
    for j in range(n):
        x[i][j] = 0

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>3</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>
for i in range(m):
    for j in range(n):
        x[i][j] = 0

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>3</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

i   j
1   2
for i in range(m):
    for j in range(n):
        x[i][j] = 0
for i in range(m):
    for j in range(n):
        x[i][j] = 0

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>3</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>
for i in range(m):
    for j in range(n):
        x[i][j] = 0

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>3</td>
<td>8</td>
</tr>
</tbody>
</table>

\( i \quad j \)

\( 2 \quad 0 \)
for i in range(m):
    for j in range(n):
        x[i][j] = 0
for i in range(m):
    for j in range(n):
        x[i][j] = 0
for i in range(m):
    for j in range(n):
        x[i][j]=0
for i in range(m):
    for j in range(n):
        x[i][j]=0
for i in range(m):
    for j in range(n):
        x[i][j] = 0

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```python
x=np.zeros((3,3))
for i in range(3):
    for j in range(3):
        x[i][j]=i
```

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 0 0</td>
<td>0 1 2</td>
<td>0 1 2</td>
</tr>
<tr>
<td>1</td>
<td>1 1 1</td>
<td>0 1 2</td>
<td>1 1 2</td>
</tr>
<tr>
<td>2</td>
<td>2 2 2</td>
<td>0 1 2</td>
<td>2 2 2</td>
</tr>
</tbody>
</table>
```python
x=np.zeros((3,3))
for i in range(3):
    for j in range(3):
        x[i][j]=j
```
```python
x = np.zeros((3, 3))
for i in range(3):
    for j in range(3):
        x[i][j] = i + j
```

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1 2</td>
<td>0 1 2</td>
<td>0 1 2</td>
</tr>
<tr>
<td>1</td>
<td>2 3</td>
<td>1 2 3</td>
<td>2 3 4</td>
</tr>
<tr>
<td>2</td>
<td>3 4</td>
<td>2 3 4</td>
<td>4 5 6</td>
</tr>
</tbody>
</table>
Plot $\sin(x)$ for $x \in [0, 2\pi]$ using numpy.

```python
import numpy as np
x = np.linspace(0, 2*np.pi, 101)
y = np.sin(x)

plt.plot(x, y, 'k-')
plt.xlim(0, 2*np.pi)
plt.ylim(-1, 1)
plt.show()
```
Modeling
Consider a cup falling from the edge of a table. Describe its path and time until it hits the ground.
Consider a cup falling from the edge of a table. Describe its path and time until it hits the ground.

Two approaches:

- Use analytical equation (if available).
- Use finite difference equation otherwise.
Use analytical equation (if available).

\[ y(t) = y_0 + v_0 t + \frac{a}{2} t^2 \]

\[ y_0 = 1 \]
\[ v_0 = 0 \]
\[ a = -9.8 \]

subject to

\[ y(t) \geq 0 \]
import numpy as np

# Parameters of simulation
n = 100  # number of data points to plot
start = 0.0 # start time, s
end = 1.0  # ending time, s
a = -9.8  # acceleration, m*s**-2

# State variable initialization
t = np.linspace(start,end,n+1) # time, s
y = 1.0 + a/2 * t**2

for i in range(1,n+1):
    if y[i] <= 0: # glass has hit the ground
        y[i] = 0
Use “finite difference” equation otherwise.
Use “finite difference” equation otherwise.

\[
\frac{dy}{dt} = v(t) \approx \frac{y^{n+1} - y^n}{t^{n+1} - t^n} \rightarrow y^{n+1} = y^n + v \left( t^{n+1} - t^n \right)
\]

\[
\frac{dv}{dt} = a \approx \frac{v^{n+1} - v^n}{t^{n+1} - t^n} \rightarrow v^{n+1} = v^n + a \left( t^{n+1} - t^n \right)
\]

\[
v^n=0 = 0 \quad y^n=0 = 1 \quad a = -9.8
\]

subject to

\[
y(t) \geq 0
\]
### Modeling

<table>
<thead>
<tr>
<th>t</th>
<th>0.0</th>
<th>0.1</th>
<th>...</th>
<th>i-1</th>
<th>i</th>
<th>i+1</th>
<th>...</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>1.0</td>
<td>0.9</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>v</td>
<td>0.0</td>
<td>0.1</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>
import numpy as np

# Parameters of simulation
n = 100  # number of data points to plot
start = 0.0  # start time, s
end = 1.0  # ending time, s
a = -9.8  # acceleration, m*s**-2

# State variable initialization
t = np.linspace(start, end, n+1)  # time, s
y = np.zeros(n+1)  # height, m
v = np.zeros(n+1)  # velocity, m*s**-1
y[0] = 1.0  # initial condition, m

for i in range(1, n+1):
    v[i] = v[i-1] + a * (t[i]-t[i-1])
    y[i] = y[i-1] + v[i-1] * (t[i]-t[i-1])

    if y[i] <= 0:  # glass has hit the ground
        v[i] = 0
        y[i] = 0
How would you make the cup bounce?
How would you include lateral motion?
Next steps
exam2 this week, 2/28–3/2
hw05 will be assigned next week
Read for the next class