Final Lectures

• Which of the following would you prefer?
  a) Fundamentals of MATLAB
  b) Advanced Python
  c) Python web programming
What code produces this error?

a) \texttt{x=1}
   \hspace{1cm} \texttt{y=x*2}

b) \texttt{x=0}
   \hspace{1cm} \texttt{y+=1}

c) \texttt{x="ABCD"}
   \hspace{1cm} \texttt{y=x[2]}
SyntaxError: invalid syntax

What code produces this error?

a) if x < "HAPPY":
    print(x.lower())[1]

b) if x in "ABCD":
    print("E"+x[0])

c) if x = (1,2,3):
    print(x[2]+1)
Continue

- Skips to the next iteration of a loop
- Useful for filtering out unwanted data

```python
for i in range(100):
    if i % 2 == 0:
        continue
    print(i)
```
```python
x = []
# range with 1 argument starts at 0
for i in range(100):
    if i < 95:
        continue
    x.append(i)
```

a) [95, 96, 97, 98, 99, 100]
b) [96, 97, 98, 99]
c) [95, 96, 97, 98, 99]
d) [96, 97, 98, 99, 100]
Exceptions

• Represent computation reaching an exceptional (unexpected or unusual) state
• Exceptions are “thrown” when we reach the state `print(x)`
• If exception is not caught (or handled) Python will print a trace
  – list of lines of code that were running
infile = open("Batting.csv")
reader = DictReader(infile)
rbis = {}
for row in reader:
    player = row["playerID"]
rbi = int(row["HR"])
if player not in rbis:
    rbis[player] = rbi
else:
    rbis[player] += rbi
TypeError: Can't convert 'int' object to str implicitly

What code throws this exception?

a) 1 + (1,2)

b) "3" + len("3")

c) 5 + ''.join([0,3])
IndexError: list index out of range

What code throws this exception?

a) x = “ABCD” + “E”
   x[5]

b) x = [1,2]
   x[2]

c) x = {1:2, 2:3}
   y = x[2]
TypeError: can only concatenate tuple (not "int") to tuple

What code throws this exception?

a) for i in zip(“ABC”):
   print(i[0])

b) for i in range(“ABC”):
   print(i+1)

a) for i in enumerate(“ABC”):
   print(i+1)
Handling Exceptions

- Exceptions can be caught using the `try/except` structure

```python
try:
    a=[1,2]
    print(a[2])
except:
    print("Oh no! An error!")
```
Throwing Exceptions

• Exceptions can be thrown with the “raise” structure

    raise Exception("Don’t do that!")
Course Summary

1. Python fundamentals
2. Writing programs
3. Data wrangling
4. Data visualization
5. Simulation
6. Random processes
7. Optimization
Numpy

- Module for Python to extend its numerical capabilities
- Designed for more efficient computation
- Designed for manipulating arrays and matrices

```python
import numpy as np
```
Arrays

• Numpy arrays are similar to lists:
  – Represent a collection of items
  – Can be indexed
• Numpy arrays are different than lists:
  – Fixed size
  – All elements have the same type
  – Can do operations on all elements
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])
Data type

- Many possible types in numpy
  - Boolean
  - integers (8, 16, 32, 64 bits)
  - floats (16, 32, and 64 bits)
  - complex (64 and 128 bits)

```python
a=[3,2,4]
x=np.array(a,dtype=np.float64)
x.dtype
```
**arange**

- Returns array over a range (like list range)
  - Argument 1: Start
  - Argument 2: End
  - Argument 3: Step size

```python
x = np.arange(10, 25, 5.0)
len(x)
```
linspace

- Returns array of evenly spaced values
  - Argument 1: start of range
  - Argument 2: end of range
  - Argument 3: number of points in range

```python
x = np.linspace(0, 1, 100)
y = x**2
plt.plot(x, y, 'g--')
```
zeros

- Returns array of zeros
  - Argument 1: the number of zeros

```python
x = np.zeros(100)
```
```python
x.dtype
```
```python
x.size
```
Why use numpy?

• Extremely powerful!

```python
x=np.linspace(0,2*np.pi,100)
y=np.sin(x)
plt.plot(x,y,'g--')
```
2D ARRAYS
Multidimensional Arrays

• Arrays can be *multidimensional*
• Let’s make a 3x2 array
  – 2 dimensional array
  – 3 rows, 2 columns

\[
a = \begin{bmatrix} 
1 & 2 \\
3 & 4 \\
5 & 6 
\end{bmatrix}
\]  # List of  # lists!

\[
b = \text{np.array}(a)
\]
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]])`
### 2D Arrays

#### 3 rows

<table>
<thead>
<tr>
<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>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>3</td>
<td>8</td>
</tr>
</tbody>
</table>
2D indexing

• We must specify both the row *and* column number to retrieve an element
• Row is first, then column:
  \( a[r][c] \)
### 2D Arrays

The 2D array `a[1][2]` is illustrated with the following values:

<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>

The value at `a[1][2]` is highlighted as 2.
<table>
<thead>
<tr>
<th>1</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>

How can we index 5?

- a) $a[1][2]$
- b) $a[2][1]$
- c) $a[1][1]$
- d) $a[2][2]$
zeros

- Returns array of zeros
  - Argument 1: a tuple/list of dimensions
  
x = np.zeros((10, 10))

x.shape
Looping over 2D arrays

```python
x = np.zeros((3, 3))
for i in range(3):
    print x[i]
```
Looping over 2D arrays

```python
x=np.zeros((3,3))
for i in range(3):
    x[i][0]=1
    x[i][1]=2
    x[i][2]=3
print x
```
Looping over 2D arrays

```python
x = np.zeros((3, 3))
for i in range(3):  # for each row
    x[i][0] = 1
    x[i][1] = 2  # columns in the row
    x[i][2] = 3
print(x)
```
Looping over 2D arrays

```python
x[i][0] = 1
x[i][1] = 2
x[i][2] = 3
for j in range(3):
    x[i][j] = j + 1
```
Looping over 2D arrays

```python
x=np.zeros((3,3))
for i in range(3):
    x[i][0]=1
    x[i][1]=2
    x[i][2]=3
    for j in range(3):
        x[i][j]=j+1
```
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
```
The matrix is of size $m=3 \times n=4$. It is given by:

\[
\begin{pmatrix}
5 & 2 & 1 & 2 \\
4 & 3 & 2 & 1 \\
9 & 4 & 3 & 8 \\
\end{pmatrix}
\]
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
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>0</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<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>

i  j
---  ---
0  1
```python
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>0</th>
<th>0</th>
<th>0</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>3</td>
<td>8</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>
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>
</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 j
for i in range(m):
    for j in range(n):
        x[i][j]=0

\[
\begin{array}{cccc}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
9 & 4 & 3 & 8 \\
\end{array}
\]
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>
</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>4</td>
<td>3</td>
<td>8</td>
</tr>
</tbody>
</table>

i  j
2  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>
</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>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

i 2
j 3
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>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<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
```

A

```
0 0 0
1 1 1
2 2 2
```

B

```
0 1 2
0 1 2
0 1 2
```

C

```
0 1 2
1 1 1
2 2 2
```
x = np.zeros((3, 3))
for i in range(3):
    for j in range(3):
        x[i][j] = j

A

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

B

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

C

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

A =
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

B =
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

C =
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>
a={"D":2,"O":5,"G":3}
for k in "DOGGY":
    print a[k]

What error will this code produce?
a) SyntaxError: invalid syntax
b) KeyError: 'Y'
c) TypeError: list indices must be integers, not str
d) There is no error.
a={"D":2,"O":5,"G":3}
for k in "DOGGY":
    print a[k]

What will this code output before it crashes?
a) "D" "O" "G" and "G"
b) 2 5 3 3
c) None None None None

d) Nothing at all
```python
x = []
for c in "ABCDEFG":
    if c < "D":  
        continue
    x.append(c)
```

Options:

a) ["A", "B", "C", "D", "E", "F", "G"]
b) ["A", "B", "C"]
c) ["D", "E", "F", "G"]
d) []
How can we index 5?

a) a[1][2]
b) a[2][0]
c) a[1][1]
d) a[2][2]
for i in range(m):
    for j in range(n):
        x[i][j] = 0
x = np.zeros((3, 3))
for i in range(3):
    for j in range(3):
        x[i][j] = i

A = 
| 0 0 0 |
| 1 1 1 |
| 2 2 2 |

B = 
| 0 1 2 |
| 0 1 2 |
| 0 1 2 |

C = 
| 0 1 2 |
| 1 1 2 |
| 2 2 2 |
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
```
BASIC SIMULATION
Example

• A kitten knocks a cup off of a 1-meter high table. How long until it hits the ground?
• \( g = -9.8 \text{m/s}^2 \)
• \( v_0 = 0 \text{m/s}, \ y_0 = 1 \text{m} \)
• \( v_{t+1} = v_t + g \Delta t \)
• \( y_{t+1} = y_t + v_t \Delta t \)
• \( \Delta t = ? \)
import numpy as np

# Parameters of simulation
n=100           # number of data points to plot
start=0.0       # start time of simulation
end=1.0         # ending time of simulation
g=-9.8          # acceleration of gravity

# State variable initialization

t=np.linspace(start,end,n+1)    # time in seconds
y=np.zeros(n+1)                 # height in meters
v=np.zeros(n+1)                 # velocity in m/s (v0=0m/s)
y[0]=1.0                        # initialize height to 1m

for i in range(1,n+1):
    v[i]=v[i-1]+g*(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
State variables

• A collection of variables describing the current state of the system.
• Describe all information we are interested in.
• Describe all information needed to determine the future state.
Model pseudocode

set constants
state=initial_state
while not simulation_finished:
    state=update(state)
import numpy as np

# Parameters of simulation
n=100           # number of data points to plot
start=0.0       # start time of simulation
end=1.0         # ending time of simulation
g=-9.8          # acceleration of gravity

# State variable initialization
t=np.linspace(start,end,n+1)    # time in seconds
y=np.zeros(n+1)                 # height in meters
v=np.zeros(n+1)                 # velocity in m/s (v0=0m/s)
y[0]=1.0                        # initialize height to 1m

for i in range(1,n+1):
    v[i]=v[i-1]+g*(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
Time step size

• How much time passes between iterations of our simulation.

• What is $\Delta t$?

• Determines the resolution/fidelity of our model.

• Smaller step size means more accurate reproduction of reality.

• Smaller step size means longer run time.
v=0.0
y=1.0
g=-9.8
t=0
dt=???

while(y>0.0):
    t+=dt
    y+=v*dt
    v+=g*dt

Which of these values will produce the *fastest* simulation?

a) 1
b) .1
c) .01
d) .001
Multiple Simulations

• Sometimes, we will need to run multiple versions of a simulation
  – e.g. testing multiple parameters

• Create a 2D array for state variables
  – Each row tracks a different simulation (angle)
  – Each column tracks one step in simulation
# Parameters of simulation

n=1000          # number of data points to plot
m=20            # number of balls to drop
start=0.0       # start time of simulation
end=2.0         # ending time of simulation
g=-9.8          # acceleration of gravity

# State variable initialization

```
t=np.linspace(start,end,n+1)    # time in seconds
y=np.zeros((m,n+1),dtype=np.float64)            # height in meters
v=np.zeros((m,n+1),dtype=np.float64)            # velocity in m/s

for i in range(m):
    y[i][0]=i+1

for i in range(m): # glass number
    for j in range(1,n+1): # time number
        if y[i][j-1]>0:
            y[i][j]=y[i][j-1]+v[i][j-1]*(t[j]-t[j-1])
            v[i][j]=v[i][j-1]+g*(t[j]-t[j-1])
        else:
            y[i][j]=0
            v[i][j]=0
```