Announcements

- My office hours have changed: Now Wed., 10:00-12:00
- No office hours this Friday, Oct. 12
- Read chapter 10, encodings
- Work on project 3
Data Structures

• So far, we have seen *native* data structures:
  • Simple values: int, float, Boolean
  • Sequences:
    • Range, string, list
    • Tuple, dictionary (chapter 11)

• There are many more useful data structures, not part of the Python language

• How can we get and use those data structures?
Encoded data structures

- Our first encoding: matrices
- What is a matrix?
  \[ A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \]
- Python does not have this data structure natively, so we need to encode it
- Two tasks are needed
  - We need to store the matrix entries
  - We need to find and access them
Matrix indexing

- Matrix
  \[ A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \]

- “Native indexing,” familiar from mathematics:

- Python encoded indexing:
  - Could mimic native encoding, but best done zero-up:
  - So, how does the Python encoded indexing work?
Matrix encoding

- Matrix

\[ A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \]

- Encode matrix as a list:
  - \( A = [1, 2, 3, 4, 5, 6] \)

- Python encoded indexing requires a mapping:
  - All elements of \( A[0][k] \) are first, as \( A[k] \)
  - All elements of \( A[1][k] \) come next, as \( A[3+k] \)
  - 3 is the row length

- In general, element \( A[i][k] \) is in position \( [i*r+k] \), where \( r \) is the row length
Does this work?

- We lose a bit of information in this encoding
  - Which numbers correspond to which row
- We must *explicitly* keep track of rows through a row length variable

\[
\mathcal{B} = \begin{pmatrix} 1 & 0 & 0 \\ 0.5 & 3 & 4 \\ -1 & -3 & 6 \end{pmatrix}
\]

\[
\mathbf{B} = [1, 0, 0, 0.5, 3, 4, -1, -3, 6]
\]

\[
\text{rowLength} = 3
\]

\[
\mathbf{B}[\text{rowLength}*y + x]
\]
Let’s check

\[ B = \begin{pmatrix}
1 & 0 & 0 \\
0.5 & 3 & 4 \\
-1 & -3 & 6 \\
\end{pmatrix} \]

\[ B = [1, 0, 0, 0.5, 3, 4, -1, -3, 6] \]

rowLength = 3

\[ B[\text{rowLength}\times y + x] \]

\[ \begin{align*}
x &= 0 \\
y &= 0 \\
B[3\times0 + 0] & \end{align*} \]

\[ \begin{align*}
x &= 1 \\
y &= 1 \\
B[3\times1 + 1] & \end{align*} \]

\[ \begin{align*}
x &= 2 \\
y &= 1 \\
B[3\times1 + 2] & \end{align*} \]
CQ: which mapping?

- \( A = \begin{pmatrix} 0 & 1 & 2 \\ 5 & 4 & 3 \end{pmatrix} \) stored as list \( A = [0,1,2,5,4,3] \), indexed zero-up: \( A[1][1] = 4 \)

```python
def get_Elt_1(i, k, A):
    p = i*3 + k
    return A[p]

def get_Elt_2(i, k, A):
    p = k*3 + i
    return A[p]

def get_Elt_3(i, k, A):
    p = i*3 + k - 1
    return A[p]
```

A) get_Elt_1
B) get_Elt_2
C) get_Elt_3
Another way to encode a Matrix

- Lets take a look at our example matrix

\[
\begin{bmatrix}
1 & 0 & 0 \\
0.5 & 3 & 4 \\
-1 & -3 & 6
\end{bmatrix}
\]

- What about this?
  - \( B = [[1, 0, 0], [0.5, 3, 4], [-1, -3, 6]] \)
Better matrix encoding

- Matrix
  \[ A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \]

- Encode matrix as a list of lists, each row a list:
  - \[ A = \left[ \begin{array}{lll} 1 & 2 & 3 \\ 4 & 5 & 6 \end{array} \right] \]

- Python encoded indexing is now:
  - All elements of \( A[0][k] \) are the first row
  - All elements of \( A[1][k] \) are the second row
  - The **row length** is reflected in the encoding structure

- In general, element \( A[i][k] \) is what we want, but with zero indexing:
  \[ A[i, k] = A[i-1][k-1] \]
Why is this important?

- We can now write code that more closely resembles mathematical notation
  - i.e., we can use \( x \) and \( y \) to index into our matrix

```python
B = [[1, 0, 0], [0.5, 3, 4], [-1, -3, 6]]
for x in range(3):
    for y in range(3):
        print (B[x][y])
```
How do we get simple matrices programmed?

- Recall: we can use the "*" to create a multi element sequence:
  - 6 * [0] results in a sequence of 6 0's -- [0, 0, 0, 0, 0, 0]
  - 3 * [0, 0] results in a sequence of 6 0's -- [0, 0, 0, 0, 0, 0]
  - 10 * [0, 1, 2] results in what?
What is going on under the hood?

- Python uses some algebraic conventions
  - $3 \times [0, 0]$ is short for
  - $[0, 0] + [0, 0] + [0, 0]$

- We know that “+” concatenates two sequences together
Another way to define lists

- The ‘*’ construct works for repeating the same thing:
  - 3 * [1,2] yields [1,2,1,2,1,2]

- Leveraging the `for` loop:
  - `[ <elt>  for <index> in range(<value>) ]`
  - creates a list executing the for-loop:
  - `L = [ ]`
    for k in range(<value>): L.append(<elt>)

- Example: `[ 0 for i in range(6)]` ≡ `[0]*6 and yields `[0, 0, 0, 0, 0, 0]`

- Example: `[ k for k in range(3)]` yields `[0, 1, 2]`

- What does this do: `[2*[0] for i in range(3)]`?
Defining simple matrices

- 4-by-4 all zero matrix:
  \[
  [4*[0] \text{ for } k \text{ in range}(4)]
  \]

- 5-by-5 identity matrix:
  
  ```python
  M = [5*[0] \text{ for } j \text{ in range}(5)]
  for j \text{ in range}(5):
    M[j][j] = 1
  ```
Adding two matrices

\[
M3[i][k] = M1[i][k] + M2[i][k]
\]

\[
M1 = \begin{bmatrix}
1 & 2 & 3 & 0 \\
4 & 5 & 6 & 0 \\
7 & 8 & 9 & 0
\end{bmatrix}
\]

\[
M2 = \begin{bmatrix}
2 & 4 & 6 & 0 \\
1 & 3 & 5 & 0 \\
0 & -1 & -2 & 0
\end{bmatrix}
\]

\[
M3 = \begin{bmatrix}
4 & 0 \\
4 & 0 \\
4 & 0
\end{bmatrix}
\]

for x in range(3):
    for y in range(4):
        M3[x][y] = M1[x][y] + M2[x][y]
Matrix – vector multiplication

Let A be a $3 \times 4$ matrix and V a vector of length 4. The result is a vector $W$ of length 3

$W = 3*[0]$
$V = [1,2,3,5]$
$A = [[0,1,0,5],[2,3,-1,0],[0,0,3,7]]$

for $i$ in range(3):
    for $k$ in range(4):
        $W[i]=W[i]+A[i][k]*V[k]$
Data structures

- We have constructed our first data structure!
  - As the name implies, we have given structure to the data
    - The data corresponds to the elements in the matrix
    - The structure is a list of lists
      - The structure allows us to utilize math-like notation
Homework

- Read Chapter 10 of our text (encodings)
- Work on Project 3
- If you feel not yet fluent in Python, code up some exercises or use codelab
Some points on project 3