Matrix

GITAA
Transforming careers

In this lecture

- Matrices
  - Create matrices
  - Dimensions
  - Modifying matrices
  - Accessing elements of a matrix
  - Matrix operations

Matrices

- Rectangular arrangement of numbers in rows and columns
- Rows run horizontally and columns run vertically

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}_{3x3} \qquad \begin{pmatrix} a_{11} \\ a_{21} \\ a_{31} \end{pmatrix}_{3x1} \qquad \begin{pmatrix} a_{11} & a_{12} & a_{13} \end{pmatrix}_{1x3}$$

Create a matrix

- `matrix()`- returns a matrix from an array type object or string of data
- Syntax: `numpy.matrix(data)`

```
import numpy as np
a=np.matrix("1,2,3,4;4,5,6,7;7,8,9,10")
```

```
In [17]: print(a)
[[ 1  2  3  4]
 [ 4  5  6  7]
 [ 7  8  9 10]]
```

Matrix properties

- **shape()**- returns number of rows and columns from a matrix

```
In [18]: a.shape
Out[18]: (3, 4)
```

- **shape[0]**- returns the number of rows
- **shape[1]**- returns the number of columns

```
In [20]: a.shape[0]
Out[20]: 3
```

```
In [21]: a.shape[1]
Out[21]: 4
```

- **size()**- returns the number of elements from a matrix

```
In [19]: a.size
Out[19]: 12
```

Modifying matrix using insert()

- `insert`- adds values at a given position and axis in a matrix
- Syntax: `numpy.insert(matrix,obj,values,axis)`
  - `matrix` - input matrix
  - `obj` - index position
  - `values` - matrix of values to be inserted
  - `axis` - axis along which values should be insert

Modifying matrix using insert()

- Adding the matrix '***col_new***' as a new column to ***a***
- Create a matrix

```
col_new=np.matrix("2,3,4")
```

```
In [23]: print(col_new)
[[2 3 4]]
```

```
a=np.insert(a,0,col_new,axis=1)
```

```
In [7]: print(a)
[[ 2  1  2  3  4]
 [ 3  4  5  6  7]
 [ 4  7  8  9 10]]
```

Modifying matrix using insert()

- Adding the matrix '***row_new***' as a new row to ***a***
- Create a matrix

```
row_new=np.matrix("4,5,6,7,9")
```

```
In [25]: print(row_new)
[[4 5 6 7 9]]
```

```
a=np.insert(a,0,row_new,axis=0)
```

```
In [9]: print(a)
[[ 4  5  6  7  9]
 [ 2  1  2  3  4]
 [ 3  4  5  6  7]
 [ 4  7  8  9 10]]
```

Modifying matrix using index

- Elements of matrix can be modified using index number
- Matrix ***a***

```
In [9]: print(a)
[[ 4  5  6  7  9]
 [ 2  1  2  3  4]
 [ 3  4  5  6  7]
 [ 4  7  8  9 10]]
```

- Here the value 1 should be updated to -3

```
a[1,1]=-3
```

- Print the updated matrix

```
In [15]: print(a)
[[ 4  5  6  7  9]
 [ 2 -3  2  3  4]
 [ 3  4  5  6  7]
 [ 4  7  8  9 10]]
```

Accessing elements of matrix using index

- Current matrix ***a***

```
In [15]: print(a)
[[ 4  5  6  7  9]
 [ 2 -3  2  3  4]
 [ 3  4  5  6  7]
 [ 4  7  8  9 10]]
```

- Extract elements from second row of matrix ***a***

```
In [17]: print(a[1,:])
[[ 2 -3  2  3  4]]
```

Accessing elements of matrix using index

- Extract elements from third column of matrix ***a***

```
In [20]: print(a[:,2])
[[6]
 [2]
 [5]
 [8]]
```

- Extract element with index (1,2) from ***a***

```
In [19]: print(a[1,2])
2
```

Matrix addition

- `numpy.add()`- performs elementwise addition between two matrices
- Syntax: `numpy.add(matrix_1,matrix_2)`
- Create two matrix ***A*** and ***B***

```
A = np.matrix(np.arange(0,20)).reshape(5,4)
B=np.matrix(np.arange(20,40)).reshape(5,4)
```

Matrix addition

- Print ***A*** and ***B***

```
In [32]: print(A)
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]
 [16 17 18 19]]
```

```
In [34]: print(B)
[[20 21 22 23]
 [24 25 26 27]
 [28 29 30 31]
 [32 33 34 35]
 [36 37 38 39]]
```

```
np.add(A,B)
```

```
In [35]: print(np.add(A,B))
[[20 22 24 26]
 [28 30 32 34]
 [36 38 40 42]
 [44 46 48 50]
 [52 54 56 58]]
```

Matrix subtraction

- `numpy.subtract()`- performs elementwise subtraction between two matrices
- Syntax: `numpy.subtract(matrix_1,matrix_2)`
- Consider the same matrix ***A*** and ***B***

```
A = np.matrix(np.arange(0,20)).reshape(5,4)

B=np.matrix(np.arange(20,40)).reshape(5,4)
```

Matrix subtraction

- Print ***A*** and ***B***

```
In [32]: print(A)
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]
 [16 17 18 19]]
```

```
In [34]: print(B)
[[20 21 22 23]
 [24 25 26 27]
 [28 29 30 31]
 [32 33 34 35]
 [36 37 38 39]]
```

```
np.subtract(A,B)
```

→

```
In [36]: print(np.subtract(A,B))
[[-20 -20 -20 -20]
 [-20 -20 -20 -20]
 [-20 -20 -20 -20]
 [-20 -20 -20 -20]
 [-20 -20 -20 -20]]
```

Matrix multiplication

- **numpy.dot()**- performs matrix multiplication between two matrices
- Syntax: **numpy.dot(matrix_1,matrix_2)**
- Consider the same matrix ***A*** and ***B***

```
A = np.matrix(np.arange(0,20)).reshape(5,4)

B=np.matrix(np.arange(20,40)).reshape(5,4)
```

Matrix multiplication

- Print ***A*** and ***B***

```
In [32]: print(A)
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]
 [16 17 18 19]]
```

```
In [34]: print(B)
[[20 21 22 23]
 [24 25 26 27]
 [28 29 30 31]
 [32 33 34 35]
 [36 37 38 39]]
```

For matrix multiplication, number of columns in matrix A should be equal to number of rows in matrix B

`np.dot(A,B)` →

```
In [38]: print(np.dot(A,B))
Traceback (most recent call last):

  File "<ipython-input-38-fbcbad2a0033>", line 1, in <module>
    print(np.dot(A,B))

ValueError: shapes (5,4) and (5,4) not aligned: 4 (dim 1) != 5 (dim 0)
```

Matrix multiplication

- Transpose matrix B to make it 4x5 in dimension

```
B=np.transpose(B)
```

```
np.dot(A,B)
```

→

```
In [72]: print(np.dot(A,B))
[[ 134  158  182  206  230]
 [ 478  566  654  742  830]
 [ 822  974 1126 1278 1430]
 [1166 1382 1598 1814 2030]
 [1510 1790 2070 2350 2630]]
```

- `numpy.matmul()` and `@` can also be used for matrix multiplication

Matrix multiplication

- `numpy.multiply()`- performs elementwise multiplication between two matrices
- Syntax: `numpy.multiply(matrix_1,matrix_2)`

```
In [32]: print(A)
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]
```

```
np.multiply(A,B)
```

```
In [34]: print(B)
[[20 21 22 23]
 [24
 [28
 [32
```

```
In [77]: print(np.multiply(A,B))
[[  0  21  44  69]
 [ 96 125 156 189]
 [224 261 300 341]
 [384 429 476 525]
 [576 629 684 741]]
```

Matrix division

- **numpy.divide()**- performs elementwise division between two matrix
- Syntax: **numpy.divide(matrix_1,matrix_2)**
- Consider the same matrix ***A*** and ***B***

```
A = np.matrix(np.arange(0,20)).reshape(5,4)

B=np.matrix(np.arange(20,40)).reshape(5,4)
```

Matrix division

- Print ***A*** and ***B***

```
In [32]: print(A)
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]
 [16 17 18 19]]
```

```
In [34]: print(B)
[[20 21 22 23]
 [24 25 26 27]
 [28 29 30 31]
 [32 33 34 35]
 [36 37 38 39]]
```

```
np.divide(A,B)
```

```
In [39]: print(np.divide(A,B))
[[ 0.          0.04761905  0.09090909  0.13043478]
 [ 0.16666667  0.2         0.23076923  0.25925926]
 [ 0.28571429  0.31034483  0.33333333  0.35483871]
 [ 0.375       0.39393939  0.41176471  0.42857143]
 [ 0.44444444  0.45945946  0.47368421  0.48717949]]
```

Summary

- Create matrix
- Matrix properties
- Modifying matrix
- Accessing element of matrix
- Matrix operations


operation == "MIRROR_X":
mirror_mod.use_x = True
mirror_mod.use_y = False
mirror_mod.use_z = False
_operation == "MIRROR_Y":
mirror_mod.use_x = False
mirror_mod.use_y = True
mirror_mod.use_z = False
_operation == "MIRROR_Z":
mirror_mod.use_x = False
mirror_mod.use_y = False
mirror_mod.use_z = True
selection at the end -add
_ob.select= 1
ier_ob.select=1
ntext.scene.objects.activ
("Selected" + str(modifier
irror_ob.select = 0
= bpy.context.selected_ob
ata.objects[one.name].se
int("please select exactl

THANK YOU