Lab 01

Lab 01

- Numpy
- Image manipulation

Using numpy

- import numpy as np

Arrays

- Arrays represent matrices in numpy

```
a = np.array([1, 2, 3]) # Create a rank 1 array
print(type(a), a.shape, a[0], a[1], a[2])
a[0] = 5 # Change an element of the array
print(a)

b = np.array([[1,2,3],[4,5,6]]) # Create a rank 2 array
print(type(b), b.shape)
print(b)
print(b[0, 0], b[0, 1], b[1, 0])
```

```
<class 'numpy.ndarray'> (3,) 1 2 3
[5 2 3]
<class 'numpy.ndarray'> (2, 3)
[[1 2 3]
 [4 5 6]]
1 2 4
```

initialized arrays

```
a = np.zeros((2,2))  # Create an array of all zeros
print(a)
```

```
[[ 0.  0.]
 [ 0.  0.]]
```

```
b = np.ones((1,2))   # Create an array of all ones
print(b)
```

```
[[ 1.  1.]]
```

initialized arrays

```
c = np.full((2,2), 7) # Create a constant array
print(c)
```

```
[[7 7]
 [7 7]]
```

```
d = np.eye(2)         # Create a 2x2 identity matrix
print(d)
```

```
[[ 1.  0.]
 [ 0.  1.]]
```

```
e = np.random.random((2,2)) # Create an array filled with random values
print(e)
```

```
[[ 0.72534879  0.18028294]
 [ 0.72548017  0.2593257 ]]
```

Array indexing : slicing

- Equivalent to list slicing

```
import numpy as np

# Create the following rank 2 array with shape (3, 4)
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

# Use slicing to pull out the subarray consisting of the first 2 rows
# and columns 1 and 2; b is the following array of shape (2, 2):
# [[2 3]
#  [6 7]]
b = a[:2, 1:3]
print(b)
```

```
[[2 3]
 [6 7]]
```

Array indexing : slicing

- Similar to list slicing

```
import numpy as np

# Create the following rank 2 array with shape (3, 4)
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

# Use slicing to pull out the subarray consisting of the first 2 rows
# and columns 1 and 2; b is the following array of shape (2, 2):
# [[2 3]
#  [6 7]]
b = a[:2, 1:3]
print(b)
```

```
[[2 3]
 [6 7]]
```

Array indexing : slicing

- Slice = view

```
                                    # [[ 1  2  3  4]
                                    #  [ 5  6  7  8]
                                    #  [ 9 10 11 12]]
print(a[0, 1])
b[0, 0] = 77     # b[0, 0] is the same piece of data as a[0, 1]
print(a[0, 1])
```

```
2
77
```

Array indexing : integer indexing

```
a = np.array([[1,2], [3, 4], [5, 6]])
```

```
>>> a
array([[1, 2],
       [3, 4],
       [5, 6]])
```

```
# An example of integer array indexing.
# The returned array will have shape (3,) and
print(a[[0, 1, 2], [0, 1, 0]])

# The above example of integer array indexing is equivalent to this:
print(np.array([a[0, 0], a[1, 1], a[2, 0]]))
```

```
[1 4 5]
[1 4 5]
```

Array indexing : integer indexing

- Integer indexing creates a new array

```
>>> a = np.array([[1,2], [3, 4], [5, 6]])
>>> b = a[[0,0],[1,1]]
>>> a
array([[1, 2],
       [3, 4],
       [5, 6]])
>>> b
array([2, 2])
>>> b[0] = 55
>>> b
array([55,  2])
>>> a
array([[1, 2],
       [3, 4],
       [5, 6]])
```

Array indexing : mixed indexing

```
row_r1 = a[1, :]    # Rank 1 view of the second row of a
row_r2 = a[1:2, :]  # Rank 2 view of the second row of a
row_r3 = a[[1], :]  # Rank 2 view of the second row of a
print(row_r1, row_r1.shape)             # [[ 1  2  3  4]
print(row_r2, row_r2.shape)             #  [ 5  6  7  8]
print(row_r3, row_r3.shape)             #  [ 9 10 11 12]]
```

```
[5 6 7 8] (4,)
[[5 6 7 8]] (1, 4)
[[5 6 7 8]] (1, 4)
```

Array indexing : mixed indexing

- Mixed indexing = view

```
>>> a
array([[1, 2],
       [3, 4],
       [5, 6]])
>>> b = a[0,:]
>>> b
array([1, 2])
>>> b[0] = 55
>>> a
array([[55,  2],
       [ 3,  4],
       [ 5,  6]])
```

Mutating elements

```
# Create a new array from which we will select elements
a = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
print(a)
```

```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
```

```
# Create an array of indices
b = np.array([0, 2, 0, 1])

# Select one element from each row of a using the indices in b
print(a[np.arange(4), b])  # Prints "[ 1  6  7 11]"
```

```
[ 1  6  7 11]
```

Mutating elements

```
>>> a = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
>>> print(a)
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
>>> bb = list(range(4))
>>> bb
[0, 1, 2, 3]
>>> b = [0,2,0,1]
>>> a[bb,b]
array([ 1,  6,  7, 11])
```

Mutating elements

```
# Mutate one element from each row of a using the indices in b
a[np.arange(4), b] += 10
print(a)
```

```
[[11  2  3]
 [ 4  5 16]
 [17  8  9]
 [10 21 12]]
```

Boolean indexing

```
a = np.array([[1,2], [3, 4], [5, 6]])

bool_idx = (a > 2)  # Find the elements of a that are bigger than 2;
                    # this returns a numpy array of Booleans of the same
                    # shape as a, where each slot of bool_idx tells
                    # whether that element of a is > 2.

print(bool_idx)
```

```
[[False False]
 [ True  True]
 [ True  True]]
```

Boolean indexing

```
# We use boolean array indexing to construct a rank 1 array
# consisting of the elements of a corresponding to the True values
# of bool_idx
print(a[bool_idx])

# We can do all of the above in a single concise statement:
print(a[a > 2])
```

```
[3 4 5 6]
[3 4 5 6]
```

Numpy datatypes

```
x = np.array([1, 2])  # Let numpy choose the datatype
y = np.array([1.0, 2.0])  # Let numpy choose the datatype
z = np.array([1, 2], dtype=np.int64)  # Force a particular datatype

print(x.dtype, y.dtype, z.dtype)
```

```
int64 float64 int64
```

Math : add

```
x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)

# Elementwise sum; both produce the array
print(x + y)
print(np.add(x, y))
```

```
[[  6.   8.]
 [ 10.  12.]]
[[  6.   8.]
 [ 10.  12.]]
```

Math : subtract

```
# Elementwise difference; both produce the array
print(x - y)
print(np.subtract(x, y))
```

```
[[-4. -4.]
 [-4. -4.]]
[[-4. -4.]
 [-4. -4.]]
```

Math : multiply

- Note that * is elem-wise multiplication

```
# Elementwise product; both produce the array
print(x * y)
print(np.multiply(x, y))
```

```
[[  5.  12.]
 [ 21.  32.]]
[[  5.  12.]
 [ 21.  32.]]
```

Math : divide

```
# Elementwise division; both produce the array
# [[ 0.2         0.33333333]
#  [ 0.42857143  0.5       ]]
print(x / y)
print(np.divide(x, y))
```

```
[[ 0.2         0.33333333]
 [ 0.42857143  0.5       ]]
[[ 0.2         0.33333333]
 [ 0.42857143  0.5       ]]
```

Math : inner product

```
v = np.array([9,10])
w = np.array([11, 12])

# Inner product of vectors; both produce 219
print(v.dot(w))
print(np.dot(v, w))
```

Math : matrix multiplication

```
x = np.array([[1,2],[3,4]])
y = np.array([[5,6],[7,8]])
v = np.array([9,10])
w = np.array([11, 12])
```

```
# Matrix / vector product; both produce the rank 1 array [29 67]
print(x.dot(v))
print(np.dot(x, v))

# Matrix / matrix product; both produce the rank 2 array
# [[19 22]
#  [43 50]]
print(x.dot(y))
print(np.dot(x, y))
```

Useful functions

```
x = np.array([[1,2],[3,4]])

print(np.sum(x))  # Compute sum of all elements; prints "10"
print(np.sum(x, axis=0))  # Compute sum of each column; prints "[4 6]"
print(np.sum(x, axis=1))  # Compute sum of each row; prints "[3 7]"
```

```
10
[4 6]
[3 7]
```

```
print(x)
print(x.T)
```

```
[[1 2]
 [3 4]]
[[1 3]
 [2 4]]
```

```
v = np.array([[1,2,3]])
print(v)
print(v.T)
```

```
[[1 2 3]]
[[1]
 [2]
 [3]]
```

Broadcasting

- Sometimes we work with matrices in different shapes

```
# We will add the vector v to each row of the matrix x,
# storing the result in the matrix y
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
y = np.empty_like(x)   # Create an empty matrix with the same shape as x

# Add the vector v to each row of the matrix x with an explicit loop
for i in range(4):
    y[i, :] = x[i, :] + v

print(y)
```

Broadcasting

- One way to do the same

```
vv = np.tile(v, (4, 1))  # Stack 4 copies of v on top of each other
print(vv)                 # Prints "[[1 0 1]
                          #         [1 0 1]
                          #         [1 0 1]
                          #         [1 0 1]]"
y = x + vv  # Add x and vv elementwise
print(y)
```

Broadcasting

```
# We will add the vector v to each row of the matrix x,
# storing the result in the matrix y
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
y = x + v  # Add v to each row of x using broadcasting
print(y)
```

Image manipulation

- scikit-image
  - from skimage import io
  - from skimage import color

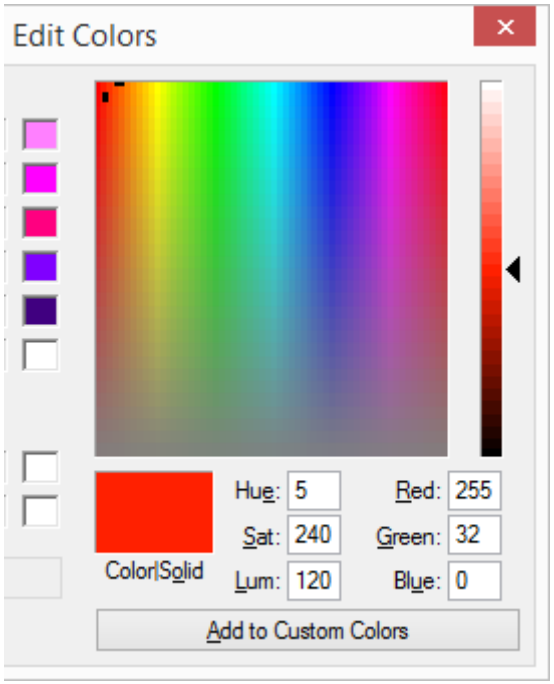
Load image

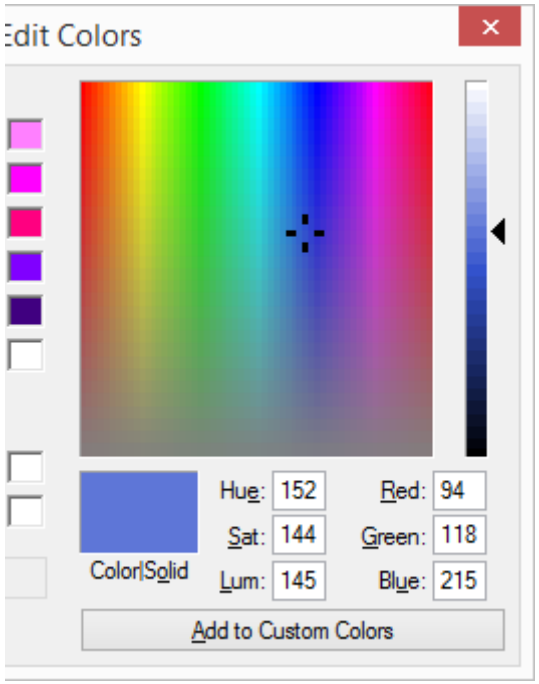
- io.imread returns numpy array of shape (height, width, 3)

```
>>> from skimage import io
>>> asdf = io.imread('image1.jpg')
>>> asdf.shape
(300, 300, 3)
```

Image representation

- numpy array of (height, width, channels)
  - (300,300,3)
  - 3 channels are for R,G,B





Save image

- io.imsave saves numpy array into a file

```
io.imsave('image11.jpg',asdf)
```

homework

- Write a function to do the followings:
  - Load an image
  - Suppress red channel in the image
  - Save the image

specification

- Write hw_[student_id].py containing a function hw1(fname_input, fname_output)
- zip to submit

Pre-check your score

- Use grader.py