IT Academy - Data Science Itinerary

Sprint 3 - Array Structure

Assignment by: Kat Weissman

Python Learning Objectives:

- Dimensions, Shapes
- Vectorization, Broadcasting
- Indexing and Masking
- Image manipulation with numpy

Recommended learning resources:

- *Numpy: https://www.w3schools.com/python/numpy/numpy_intro.asp*
- *Broadcasting: https://numpy.org/doc/stable/user/basics.broadcasting.html?highlight=broadcasting*
- *Indexing: https://numpy.org/doc/stable/user/basics.indexing.html?highlight=indexing*
- *Masking: https://numpy.org/doc/stable/reference/maskedarray.generic.html*
- *Matplotlib Image: https://matplotlib.org/stable/tutorials/introductory/images.html*

Level 1

We work on the concepts of the structure of a matrix, dimension, axes and vectorization that allows us to reduce the use of for loops in arithmetic or mathematical operations.

Exercise 1

Create a one-dimensional np.array, including at least 8 integers, data type int64. Show the size and shape of the array.

In [1]:

```
import numpy as np

#Create a one-dimensional array of 10 random integers between 0 and 100
myArray = np.random.randint(0,101,10)

#Display some of the attributes of this array, like type, dimensions, size, and shape.
```

```
print("This is my numpy array:",myArray)
print("The elements of my array are of type:", myArray.dtype)
print("My array has", myArray.ndim, "dimension.")
print("My array is of size:", myArray.size)
print("My array is of shape:", myArray.shape)
```

```
This is my numpy array: [47 29 36 61 36 79 94 77 10 43]
The elements of my array are of type: int64
My array has 1 dimension.
My array is of size: 10
My array is of shape: (10,)
```

Exercise 2

From the array in Exercise 1, calculate the mean value of the values entered and subtract the resulting average from each of the values in the array.

In [2]:

```
mean = myArray.mean()
myArray1 = myArray - mean
print(myArray1)
```

```
[ -4.2 -22.2 -15.2   9.8 -15.2  27.8  42.8  25.8 -41.2  -8.2]
```

Exercise 3

Create a two-dimensional array with a shape of 5 x 5. Extract the maximum value of the array, and the maximum values of each of its axes.

In [3]:

```
#Set the desired size of the array, then create an array of random integers at the specified size.
size = (5,5)
myArray2 = np.random.randint(0,101, size=size)
print ("My", size[0], "x", size[1], "array:")
print (myArray2)
```

```
My 5 x 5 array:
[[ 81  86  28  60  18]
 [ 13  57  36  83  89]
 [ 27 100   6  73   0]
 [ 16  71  78  24  68]
 [ 50  10  94  46  42]]
```

In [4]:

```
print ("The maximum value of my array is:", np.max(myArray2))
```

```
The maximum value of my array is: 100
```

In [5]:
```
#Axis 0 runs vertically across the rows
print("The maximum value of each column:")
print(np.amax(myArray2, 0))
```

```
The maximum value of each column:
[ 81 100  94  83  89]
```

In [6]:
```
#Axis 1 runs horizontally across the columns
print("The maximum value of each row:")
print(np.amax(myArray2, 1))
```

```
The maximum value of each row:
[ 86  89 100  78  94]
```

Level 2

Concepts of the structure of an array, broadcasting, indexing, masking.

Exercise 4

Show examples of different arrays, the fundamental rule of Broadcasting that says, "arrays can be transmitted / broadcast if their dimensions match or if one of the arrays has a size of 1."

In [7]:
```
#Arrays A & B have equivalent first dimensions, and the second dimesnion of B is 1
A = np.random.randint(0,11, size=(3,3))
B = np.random.randint(0,11, size=(3,1))
print("Array A:")
print(A)
print("Array B:")
print(B)
#Arithmetic is performed element-wise.
print("The product of A & B:")
print(A*B)
print("The sum of A & B:")
print(A+B)
```

```
Array A:
[[ 6  6  6]
 [ 7  5 10]
 [ 9  0  3]]
Array B:
```

```
[[4]
 [6]
 [7]]
The product of A & B:
[[24 24 24]
 [42 30 60]
 [63  0 21]]
The sum of A & B:
[[10 10 10]
 [13 11 16]
 [16  7 10]]
```

In [8]:

```
#Arrays A & C have equivalent second dimensions, and the first dimesnion of C is 1
C = np.random.randint(0,11, size=(1,3))
print("Array A:")
print(A)
print("Array C:")
print(C)
#Arithmetic is performed element-wise.
print("The product of A & C:")
print(A*C)
print("The sum of A & C:")
print(A+C)
```

```
Array A:
[[ 6  6  6]
 [ 7  5 10]
 [ 9  0  3]]
Array C:
[[9 1 7]]
The product of A & C:
[[54  6 42]
 [63  5 70]
 [81  0 21]]
The sum of A & C:
[[15  7 13]
 [16  6 17]
 [18  1 10]]
```

In [9]:

```
#Arrays A & D do not have equivalent second dimensions, and neither is 1, so the operation will produce an erro
D = np.random.randint(0,11, size=(3,2))
print("Array A:")
print(A)
print("Array D:")
print(D)
print("The product of A & D:")
```

```
print(A*D)
print("The sum of A & D:")
print(A+D)
```

```
Array A:
[[ 6  6  6]
 [ 7  5 10]
 [ 9  0  3]]
Array D:
[[9 7]
 [2 9]
 [5 4]]
The product of A & D:
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-9-7bafb7b9780f> in <module>
      6 print(D)
      7 print("The product of A & D:")
----> 8 print(A*D)
      9 print("The sum of A & D:")
     10 print(A+D)

ValueError: operands could not be broadcast together with shapes (3,3) (3,2)
```

The error message includes useful feedback about the reason the arrays could not be broadcast.

```
ValueError: operands could not be broadcast together with shapes (3,3) (3,2)
```

Exercise 5

Use Indexing to extract the values of a column and a row from an array. Add up their values.

In [49]:

```
print("myArray2 will be used for the following examples:")
print(myArray2)
```

```
myArray2 will be used for the following examples:
[[ 81  86  28  60  18]
 [ 13  57  36  83  89]
 [ 27 100   6  73   0]
 [ 16  71  78  24  68]
 [ 50  10  94  46  42]]
```

In [51]:

```
#indexing can be used to extract a whole row
row2 = myArray2[1]
#indexing can also be used to extract a whole column
```

```
col4 = myArray2[:,3]
print("2nd row of myArray2:")
print(row2)
print("4th column of myArray2:")
print(col4)
print("The sum of the 2nd row and the 4th column:")
print(row2 + col4)
```

```
2nd row of myArray2:
[13 57 36 83 89]
4th column of myArray2:
[60 83 73 24 46]
The sum of the 2nd row and the 4th column:
[ 73 140 109 107 135]
```

In [52]:

```
#Indexing can also be used to extract individual elements from an array.
print("1st row, 4th column:", myArray2[0,3])
print("2nd row, 1st column:", myArray2[1,0])
print("3rd row, 5th column:", myArray2[2,4])
print("4th row, 2nd column:", myArray2[3,1])
print("The sum of these values:", myArray2[0,3]+myArray2[1,0]+myArray2[2,4]+myArray2[3,1])
```

```
1st row, 4th column: 60
2nd row, 1st column: 13
3rd row, 5th column: 0
4th row, 2nd column: 71
The sum of these values: 144
```

Exercise 6

Mask an array by performing a vectorized Boolean calculation, taking each element and checking if it is evenly divided by four.

This returns a mask array in the same way as the element-wise results of a calculation.

In [11]:

```
print (myArray2)
```

```
[[ 81  86  28  60  18]
 [ 13  57  36  83  89]
 [ 27 100   6  73   0]
 [ 16  71  78  24  68]
 [ 50  10  94  46  42]]
```

In [54]:

```
#Create a boolean mask that marks elements as True if they are evenly divided by 4.
myMask = (myArray2 % 4 == 0)
```

```
print(myMask)
```

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

Exercise 7

Then use this mask to index the original number array. This causes the array to lose its original shape, reducing it to one dimension, but you still get the data you are looking for.

In [56]:
```
print(myArray2[myMask])
```

```
[ 28  60  36 100   0  16  24  68]
```

Alternatively, we can use the numpy module for Masked Arrays which will preserve the shape of the original array.

"When an element of the mask is False, the corresponding element of the associated array is valid and is said to be unmasked. When an element of the mask is True, the corresponding element of the associated array is said to be masked (invalid)." - https://numpy.org/doc/stable/reference/maskedarray.generic.html

The numpy.ma module produces a mask that has opposite boolean values of the mask created for indexing in the previous exercise.

In [64]:
```
#Use the numpy masked array method to create a masked array using the module function as the mask.
import numpy.ma as ma
npMaskedArray2 = ma.masked_array(myArray2, mask = myArray2 % 4)
print("Mask created by numpy.ma module:")
print(npMaskedArray2.mask)
print("The masked array created by the numpy.ma module:")
print(npMaskedArray2)
```

```
Mask created by numpy.ma module:
[[ True  True False False  True]
 [ True  True False  True  True]
 [ True False  True  True False]
 [False  True  True False False]
 [ True  True  True  True  True]]
The masked array created by the numpy.ma module:
[[-- -- 28 60 --]
 [-- -- 36 -- --]
 [-- 100 -- -- 0]
```

```
[16 -- -- 24 68]
[-- -- -- -- --]]
```

Level 3

Image manipulation with Matplotlib.

You will upload any image (jpg, png) with Matplotlib. note that RGB images (Red, Green, Blue) are really only widths × heights × 3 arrays (three channels Red, Green, and Blue), one for each color of int8 integers,

Manipulate these bytes and use Matplotlib again to save the modified image once you're done.

Help: Import, import matplotlib.image as mpimg. study the mpimg.imread() method

Exercise 8

Show what happens when the Green G or Blue B channel is removed. Use indexing to select the channel you want to remove.

Use the method, mpimg.imsave() of the imported library, to save the modified images and upload them to your github repository.

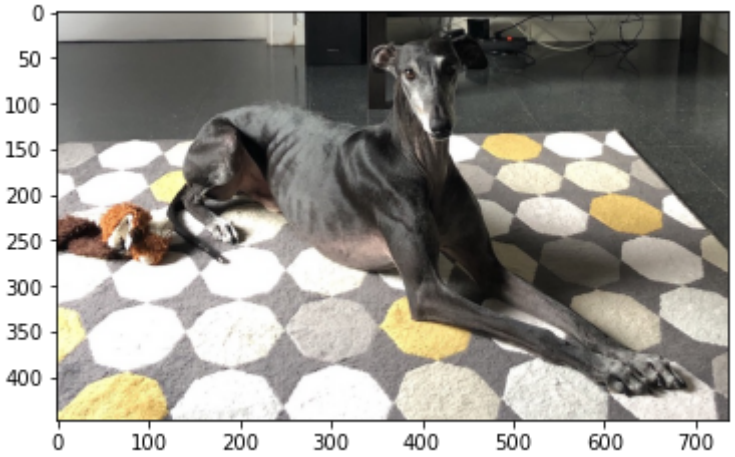
In [80]:
```
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
```

In [91]:
```
rgbImg = mpimg.imread('IMG_Logan.jpg')
```

In [92]:
```
print(rgbImg.shape)
```

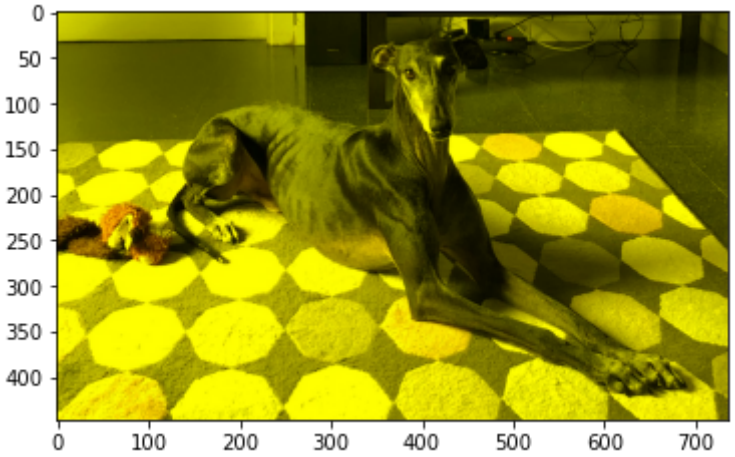
```
(447, 736, 3)
```

In [101…
```
imgplot = plt.imshow(rgbImg)
```



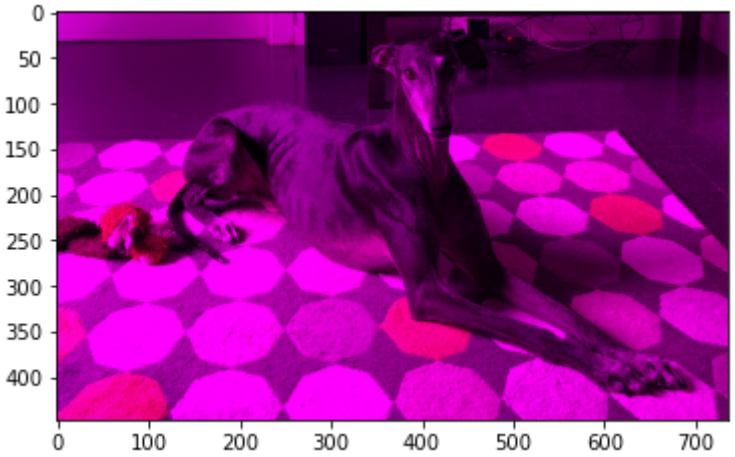
In [104…

```
#Remove the blue channel from the original image
rgImg = rgbImg.copy()
rgImg[:,:,2] = 0
rgImgPlot = plt.imshow(rgImg)
mpimg.imsave("rgImg.jpg",rgImg)
```



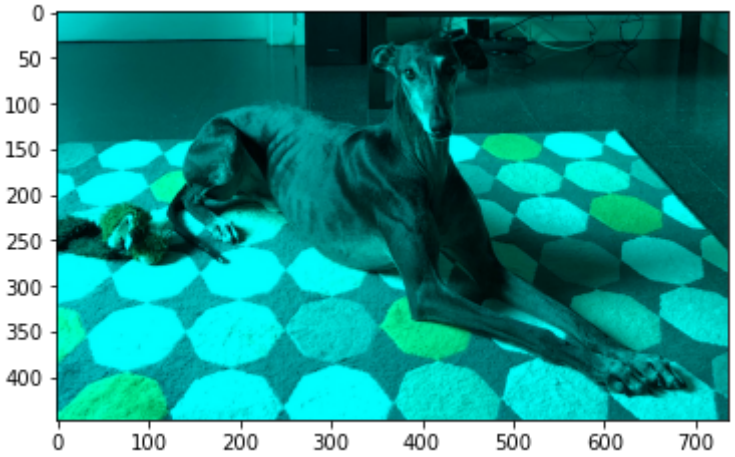
In [105…

```
#Remove the green channel from the original image
rbImg = rgbImg.copy()
rbImg[:,:,1] = 0
rbImgPlot = plt.imshow(rbImg)
mpimg.imsave("rbImg.jpg",rbImg)
```



In [106…

```
#Remove the red channel from the original image
gbImg = rgbImg.copy()
gbImg[:,:,0] = 0
gbImgPlot = plt.imshow(gbImg)
mpimg.imsave("gbImg.jpg",gbImg)
```



In [ ]: