# PROJECT 2

This project is designed with the goal of understanding how to manipulate data in python via numpy.

## 1. Table of Dispaly

|  |  |
| --- | --- |
| Data Item | Display Method |
| Data Types | Summary |
| Manipulation/Aggregation | Summary w/ Code |
| Sorting Methods | Summary w/ Code |
| Structured Arrays | Summary |
| Logical uFuncs | Summary w/code |
| Advanced Indexing | Code |
| Graphing w/matplotlib | Code |

## 2. Table of Content

* [1. Table of Dispaly](#Xb0143f333aafeb5b790b018449eb72a12a4e091)
* [2. Table of Content](#X4afda6e54e4fa9dfad905940bbfadd9e4de42dd)
* [3. Background](#X87094d5cc2003181c30bd3fb8c9e78d4fb0c530)
  + [3.1. Data Types in Python](#Xfe98627139cf48105389f2f7d068df5dcd64778)
    - [3.1.1. Python Arrays using Numpy](#X068ba5bf3337fb3756f177df55bbebb65a9b741)
      * [3.1.1.1. Additional Features of Numpy arrays](#X0a52cf2b48d843462f460a183a8f89fde2576ac)
    - [3.1.2. Structured Arrays in Python](#X66718b18e7e3c51cdf3538f3a60aa9770a479f7)
  + [3.2. Sorting Methods](#Xab505723ee4067f14eb982007d2d1b473090560)
    - [3.2.1. Paritioning](#X47e0d5b6485667dce36f8213df4961b4b28e4ab)
* [4. Boardcasting, Aggregations and Universal Functions](#X92fa39538646d306262282095755d541aa95ec8)
  + [4.1. Broadcasting](#X18025df9430566155040019ac7a26dcfc823672)
    - [4.1.1. Logical Boardcasting](#X720ad3bb20d28e154f47f3bfe0174735fb66f7e)
      * [4.1.1.1. Masks](#X6895728c2a669c9804b5a4d4d7744a5bd63c3f6)
  + [4.2. Aggregations](#Xf211ed4649eb0270096126fe5427ec2b54bf915)
* [5. Code](#X48cb430816a847b9524e41b2865aca0dbc5d680)
  + [5.1. Sameple Run](#X29c722ebfd593f66b1005edf034bbd2cb45c41f)
    - [5.1.1. Graph Output](#X1f4d496714e99aad1a557b901c6283a1ae7384a)
  + [5.2. Code](#Xdbd523c63f2980bb6b00a768ae5c781c689da1f)

## 3. Background

### 3.1. Data Types in Python

Python uses dynamic typing which means that a variable does not require a declaration of a variables type is not required.

Compare these two code segments

* JAVA
  + int x = 1  
     String y = "Hello World"
* python
  + x = 1  
     x = "Hello World"

You may have also noticed that variables in python can change the type they are. This workes because variables in python are pointers to C objects. This also means that python variables tend to have a larger overhead as each object must include its type, memeory address and size.

#### 3.1.1. Python Arrays using Numpy

Becuase of this dynamic typing python lists can become massive and cumbersom. Thus the numpy module adds the ability to design single type arrays.

Below is an example of how to use numpy arrays.

import numpy as np # Import Numpy and give it a shorthand  
  
arr1 = np.array([1,5,7,33]) # Implicitly Typed integer array  
arr2 = np.array([1.0,5,7,33]) # Implicitly Typed float array  
arr3 = np.array([1,5,7,33], dtype='float') # Explicitly Typed float array  
print(arr1)  
print(arr2)  
print(arr3)  
  
## OUTPUT  
import numpy as np # Import Numpy and give it a shorthand  
  
arr1 = np.array([1,5,7,33]) # Implicitly Typed integer array  
arr2 = np.array([1.0,5,7,33]) # Implicitly Typed float array  
arr3 = np.array([1,5,7,33], dtype='float') # Explicitly Typed float array  
print(arr1)  
print(arr2)  
print(arr3)  
  
## OUTPUT  
# [ 1 5 7 33]  
# [ 1. 5. 7. 33.]  
# [ 1. 5. 7. 33.]

A fearture that is personally peculiar to me is how numpy arrays handle nested lists. As seen in the code block below

import numpy as np  
  
#Regular Array  
arrPy = [range(i, i+5) for i in range(1,5)]  
print(arrPy)  
  
# Numpy Array  
arrNp = np.array([range(i, i+5) for i in range(1,5)])  
print(arrNp)  
  
##########  
# OUTPUT  
# [range(1, 6), range(2, 7), range(3, 8), range(4, 9)]  
# [[1 2 3 4 5]  
# [2 3 4 5 6]  
# [3 4 5 6 7]  
# [4 5 6 7 8]]  
###########

This make the creation of multidemsional array very simple but there is one word of warning. For a numpy multidemsionally array to be properly instantiated you must ensure that all rows are the same size unless you are using an array of objects.

import numpy as np  
validArr = np.array([range(i-2,i+2) for i in range(2,5)])  
print(validArr)  
invalidArr = np.array([range(int(i/2),i\*2) for i in range(2,5)])  
#ERROR ENCOUNTERED  
##################  
# OUTPUT  
#[[0 1 2 3]  
# [1 2 3 4]  
# [2 3 4 5]]  
#<ipython-input-4-4eeb350620f9>:4: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify 'dtype=object' when creating the ndarray.  
# invalidArr = np.array([range(int(i/2),i\*2) for i in range(2,5)])  
##############################################################

##### 3.1.1.1. Additional Features of Numpy arrays

Numpy also includes several utility functions when it comes to arrays. Some of these include.

* .zeros(<Array Size>, <Array Type>)
  + Creates an array of *Array Type* in *Array Size* filled with zeros
* .ones(<Array Size>, <Array Type>)
  + Creates an Array of *Array Type* in *Array Size* filled with ones
* .full(<Array Size>, <Array Value>)
  + Creates an Array of *Array Size* filled with *Array Value*
* .arrange(<Start Value>, <End Value>, <Step>)
  + Creates an array from *Start Value* to *End Value* at *Step* increments
* .linspace(<Start Value>, <End Value>, <Number of Values>)
  + Makes an array of *Number of Values* between *Start Value* and *End Value*
* .eye(<Matrix Size>)
  + Creates the Identity matrix for a square matrix of *Matrix Size*

Some of these functions can make a Multidemtional array or a normal array. These are .full, .ones, .zeros. This works if you replace *Array Size* with a tuple of your multidemtional arrays diemnsions, such as (3, 4) for a 3x4 matrix.

In addtion to auto filling an array these function also support the *dtype* value. These include bool\_,int\_,complex\_,float and uint. These each reprsent their C equvalent and the int, float and complex types also support bit values allowing your to limit the space used by your arrays.

#### 3.1.2. Structured Arrays in Python

On occasion one will need to store related data points that are related, but are not the same data type. Although numpy arrays by deault only have one data type there is a possiblity to design a structured array by manuplated the dtype value in the numpy array constructor. Below is an example.

import numpy as np  
dt = np.dtype({'names':('uName', 'uId', 'uEmail'),  
 'formats':('U10', 'i4', 'U20')})  
userData = np.zeros(3, dt)  
#Creates an Structured Array with terms 'name', 'id', and email which are a 16 charaters string, 32 bit int and 32 character string respectivly.  
userData['uName'] = ['Negative','Moon','Blue']  
userData['uId'] = [1,2,3]  
userData['uEmail'] = ['Negative@gmail.com','Moon@gmail.com','Blue@gmail.com']  
#FILLS THE ARRAY  
  
print(userData['uName']) #Print all usernames  
print(userData[0]) #Print the first entry  
print(userData[-1]['uId']) # Print the Id of the last element  
  
######  
#OUTPUT  
#['Negative' 'Moon' 'Blue']  
#('Negative', 1, 'Negative@gmail.com')  
#3  
#######

### 3.2. Sorting Methods

When working with arrays one must reqularly use sorting methods as to both organize and search for data. Although you may wish to emplement these methods yourself numpy has insluded quicksort into its module. This function can be accessed by using numpy.sort(myNpArray) which will return a sorted version of the numpy array.

This function can also sort an array along each axis of the multidiemnsional array. This can be acomplised by adding the axis parameter to the sort method. You will then assign it to the axis you wish to use starting at 0. One example of this is below.

import numpy as np  
arr1 = np.random.randint(0,100, (3,3))  
print(arr1)  
  
print('-----------------------------------------')  
# Sort along the colums  
print(np.sort(arr1, axis=0))  
print('-----------------------------------------')  
# Sort Along the Rows  
print(np.sort(arr1, axis=1))  
print('-----------------------------------------')  
# Sort whole Array by rows then columns  
print(np.sort(np.sort(arr1), axis=0))  
print('-----------------------------------------')  
# Sort whole Array by Columns then rows  
print(np.sort(np.sort(arr1, axis=0)))  
  
#################  
#OUTPUT  
#[[32 76 1]  
# [95 92 76]  
# [30 75 63]]  
#-----------------------------------------  
#[[30 75 1]  
# [32 76 63]  
# [95 92 76]]  
#-----------------------------------------  
#[[ 1 32 76]  
# [76 92 95]  
# [30 63 75]]  
#-----------------------------------------  
#[[ 1 32 75]  
# [30 63 76]  
# [76 92 95]]  
#-----------------------------------------  
#[[ 1 30 75]  
# [32 63 76]  
# [76 92 95]]

In addtion to sorting an array numpy included the .argsort() funciton. This fuynction returns an array filled with the indecies of the array in sorted order. For example

import numpy as np  
arr1 = np.array([3,2,1])  
print(np.argsort(arr1))  
#######  
#OUTPUT  
#[2,1,0]

#### 3.2.1. Paritioning

In adition to sorting numpy includes a method to search for the smallest values in an array. This method is called numpy.partition(<array>, <number of values>). This function works identically to sort and the function numpy.argpartition(<array>, <number of values>) works similarly to numpy.argsort().

## 4. Boardcasting, Aggregations and Universal Functions

Although loops are useful when operation on arrays, pythons loops tend to slow. Therefore numpy added the ability to do large operations over an array. These methods include Univeral Function, Broadcasting and Aggregations. Some of these are rather simple such as Univeral Functions which are just the python and numpy equvalent functions. Others are more complex and are explored below.

### 4.1. Broadcasting

The common uFuncs are actually a specal case of Broadcasting in which you have one array and one single element. With broadcaseing you can manipulate one array of shape (n,n) and another array of shape (k,k) and these values do not neccesarily have to be the same. Forinstance

import numpy as np  
  
arr1 = np.arange(3)  
arr2 = np.ones(3))[:, np.newaxis] # Make a columns of ones  
print(arr1+ arr2)  
####  
#OUTPUT  
#[0 1 2]  
# [[1.]  
# [1.]  
# [1.]]  
#[[1. 2. 3.]  
# [1. 2. 3.]  
# [1. 2. 3.]]

Although arrays are padded or streched when there is a diffence of dimensions at least one of the dimensions must match or not be used. This simply means that if you have two three dimensional array with compltly diffent values board casing will not work.

#### 4.1.1. Logical Boardcasting

Boardcasting can also work with logic functions and will return an array of boolans rather than an an array of numbers. This comes in handy when seeking to count values as python treats all non zero values as True and only Zero as false. There fore when coupled with np.count\_nonzero() booleans expressions become rather powerful. There is also the equvaluent to an AND and OR statment for arrays with Boolean arrays call .all() and .any() which do just as their name suggets.

##### 4.1.1.1. Masks

Boolan arrays can also be used masks which return a smaller array based on some logical expression. This technique will be evalued in the following example.

### 4.2. Aggregations

Aggregation is a specal addition to the numpy uFuncs on an array. Some of these include .reduce() which will recusivly conduct your operation on the whole array and return a single value (such as finding the sum of the array), .accumulate() which conducts the operations through the array and returns an array with each step (like a running total on an array). Although .reduce() and .accumlate() are usfult numpy arrays also have a suite of statistical functions that are incredibly usfult and productive. Below is a example of many of the numpy array aggregations.

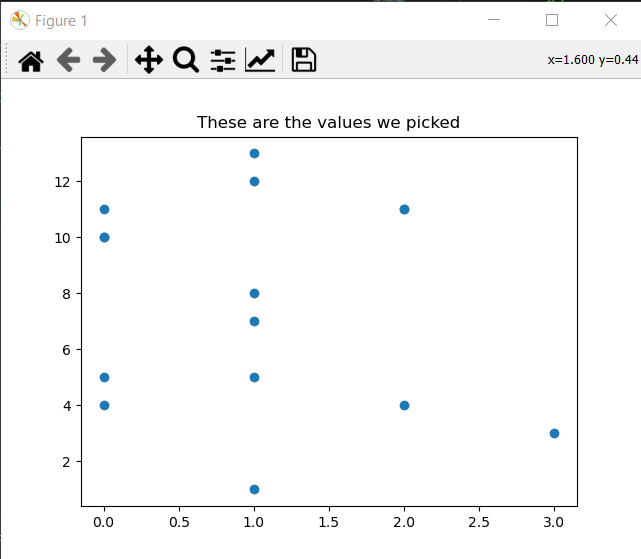
import numpy as np  
arr1 = np.random.random((5,5))  
arr2 = np.random.random(5)  
  
print(arr1,"\n",arr2)  
#SUM  
print(np.add.reduce(arr1))  
print(np.add.accumulate(arr2))  
print(arr1.sum(axis=1))  
print(arr1.sum())  
arr2 += 1 # uFun to add 1 to all values of arr2 and reassign  
#POWER  
print(np.power.accumulate(arr2))  
# Stand Deviation and Varience  
print(arr1.std())  
print(arr2.var())  
# Index of max/min  
print(arr1.argmax()) # Is to find the mas of each sub array as this will print rowcolumne with no space  
print(arr1.argmin())  
#####  
# Sample OUTPUT  
#]  
  
  
  
  
import numpy as np  
arr1 = np.random.random((5,5))  
arr2 = np.random.random(5)  
  
print(arr1,"\n",arr2)  
#SUM  
print(np.add.reduce(arr1))  
print(np.add.accumulate(arr2))  
print(arr1.sum(axis=1))  
print(arr1.sum())  
arr2 += 1 # uFun to add 1 to all values of arr2 and reassign  
#POWER  
print(np.power.accumulate(arr2))  
# Stand Deviation and Varience  
print(arr1.std())  
print(arr2.var())  
# Index of max/min  
print(arr1.argmax()) # Is to find the mas of each sub array as this will print rowcolumne with no space  
print(arr1.argmin())  
#[[0.67336848 0.17178476 0.94030908 0.15257246 0.74257853]  
# [0.20917456 0.91132532 0.00736586 0.58191279 0.16805195]  
# [0.27578636 0.41687174 0.61413864 0.53289707 0.02307563]  
# [0.52531865 0.19114826 0.91523491 0.25310389 0.538879 ]  
# [0.25670329 0.6615298 0.52004819 0.87967821 0.60668515]]  
# [0.02603621 0.2391786 0.4240816 0.73169064 0.2876162 ]  
#[1.94035134 2.35265989 2.99709668 2.40016442 2.07927027]  
#[0.02603621 0.26521481 0.68929641 1.42098705 1.70860325]  
#[2.68061332 1.87783048 1.86276944 2.42368471 2.92464464]  
#11.769542595298107  
#[1.02603621 1.03236332 1.04640234 1.08171302 1.1064282 ]  
#0.2810060427591307  
#0.05439174997540538  
#2  
#7

## 5. Code Project

### 5.1. Sameple Run

This Project was produced by: Paul A. Pace  
FOR: EGR491 Project 2 Summer 21 Mercer University  
How many rows for your array5  
How many columns for your array15  
This is your array:  
[[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]  
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]  
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]  
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]  
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]]  
applying random value  
Evaluating your array  
[[60. 57. 73. 72. 78. 94. 40. 20. 72. 78. 62. 46. 26. 97. 93.]  
 [ 7. 50. 35. 63. 42. 92. 77. 40. 74. 91. 9. 4. 8. 60. 35.]  
 [70. 22. 44. 45. 98. 2. 7. 15. 8. 23. 90. 46. 56. 48. 39.]  
 [25. 89. 77. 90. 86. 5. 82. 40. 22. 38. 94. 49. 27. 25. 51.]  
 [73. 53. 48. 32. 1. 8. 12. 93. 22. 44. 96. 73. 55. 43. 80.]]  
These values are over 50  
[[ True True True True True True False False True True True False  
 False True True]  
 [False False False True False True True False True True False False  
 False True False]  
 [ True False False False True False False False False False True False  
 True False False]  
 [False True True True True False True False False False True False  
 False False True]  
 [ True True False False False False False True False False True True  
 True False True]]  
The average value above 50 is 77.11428571428571  
It looks a little sloppy lets fix that:  
The sum of each column is:  
[ 32. 70. 93. 117. 173. 199. 225. 255. 271. 311. 344. 367. 409. 458.  
 477.]  
The Average Column Size is: 760.2  
Lets do some more:  
98.0 :MAX  
1.0 :MIN  
-------------------  
now let us do some boardcasting  
15  
(15,)  
[[-20. -25. -38. -43. -53. -55. -56. -65. -64. -64. -68. -67. -81. -81.  
 -83.]  
 [ -4. -6. -6. -6. -31. -30. -34. -35. -42. -51. -53. -63. -65. -78.  
 -78.]  
 [ -2. -6. -6. -12. -18. -18. -33. -37. -37. -37. -38. -45. -58. -77.  
 -84.]  
 [ -5. -21. -23. -22. -23. -33. -34. -42. -43. -68. -72. -75. -77. -77.  
 -80.]  
 [ -1. -7. -10. -19. -28. -38. -38. -41. -45. -46. -63. -62. -68. -80.  
 -82.]]  
[[ 40. 51. 78. 89. 110. 115. 118. 137. 136. 137. 146. 145. 174. 175.  
 180.]  
 [ 8. 13. 14. 15. 66. 65. 74. 77. 92. 111. 116. 137. 142. 169.  
 170.]  
 [ 4. 13. 14. 27. 40. 41. 72. 81. 82. 83. 86. 101. 128. 167.  
 182.]  
 [ 10. 43. 48. 47. 50. 71. 74. 91. 94. 145. 154. 161. 166. 167.  
 174.]  
 [ 2. 15. 22. 41. 60. 81. 82. 89. 98. 101. 136. 135. 148. 173.  
 178.]]  
Some Advaced Indexing  
[0 2 1 1 1 1 1 0 0 1 0 3 0 2 2]  
 [10 4 7 13 1 12 8 4 5 5 11 3 10 11 11]  
The values at these locations are:  
[78. 22. 42. 91. 7. 77. 50. 57. 60. 35. 78. 25. 78. 56. 56.]  
And just for kicks lets make a graph:  
We will print arr3 vs arr4

#### 5.1.1. Graph Output



GRAPH

### 5.2. Code

# To add a new cell, type '# %%'  
# To add a new markdown cell, type '# %% [markdown]'  
# %% [markdown]  
# # Project Goal  
#  
# Show multiple methods and function on an set of arrays including data analysis.  
  
# %%  
import numpy as np  
  
  
# %%  
def welcome():  
 print("This Project was produced by: Paul A. Pace")  
 print("FOR: EGR491 Project 2 Summer 21 Mercer University")  
  
  
# %%  
def promt4array():  
 num1 = input("How many rows for your array")  
 num2 = input("How many columns for your array")  
 return (int(num1),int(num2))  
  
  
# %%  
welcome()  
arr1 = np.ones(promt4array())  
  
  
# %%  
print("This is your array: ")  
print(arr1)  
print("applying random value")  
arr1 += np.random.randint(0,100, arr1.shape)  
  
  
# %%  
print("Evaluating your array")  
print(arr1)  
print("These values are over 50")  
mskArr = arr1 > 50  
print(mskArr)  
  
print("The average value above 50 is ", arr1[mskArr].mean())  
  
print("It looks a little sloppy lets fix that: ")  
arr1 = np.sort(arr1, axis=1)  
  
print("The sum of each column is: ")  
print(arr1.sum(axis=0))  
print("The Average Column Size is: ", end=' ')  
print(arr1.sum(axis=1).mean())  
  
  
  
# %%  
print("Lets do some more: ")  
print(arr1.max(), ":MAX")  
print(arr1.min(), ":MIN")  
  
  
# %%  
print("-------------------")  
print("now let us do some boardcasting")  
x = arr1.shape[1]  
print(x)  
arrTmp = np.array(np.arange(x))  
print(arrTmp.shape)  
arr2 = arrTmp - arr1  
print(arr2)  
print(arr1 - arr2) # This will print your column numbers  
  
  
# %%  
print("Some Advaced Indexing")  
# We must get the smallest of either rows of columns  
x = arr1.shape[0]  
if x < arr1.shape[1]:  
 x = arr1.shape[1]  
  
arr3 = np.random.randint(0, arr1.shape[0]-1, x) #Generate your Rows  
arr4 = np.random.randint(0, arr1.shape[1]-1, x) # Generate your columns  
arr3 += np.zeros(arr4.shape, dtype='int32') # You must set the type to int32 and have the same number of values  
arr4 += np.zeros(arr3.shape, dtype='int32')  
print(arr3, "\n", arr4)  
print("The values at these locations are: ")  
print(arr1[arr3,arr4])  
  
  
# %%  
print("And just for kicks lets make a graph:")  
print("We will print arr3 vs arr4")  
import matplotlib.pyplot as plt  
plt.scatter(arr3, arr4)  
plt.title("These are the values we picked")  
plt.show()