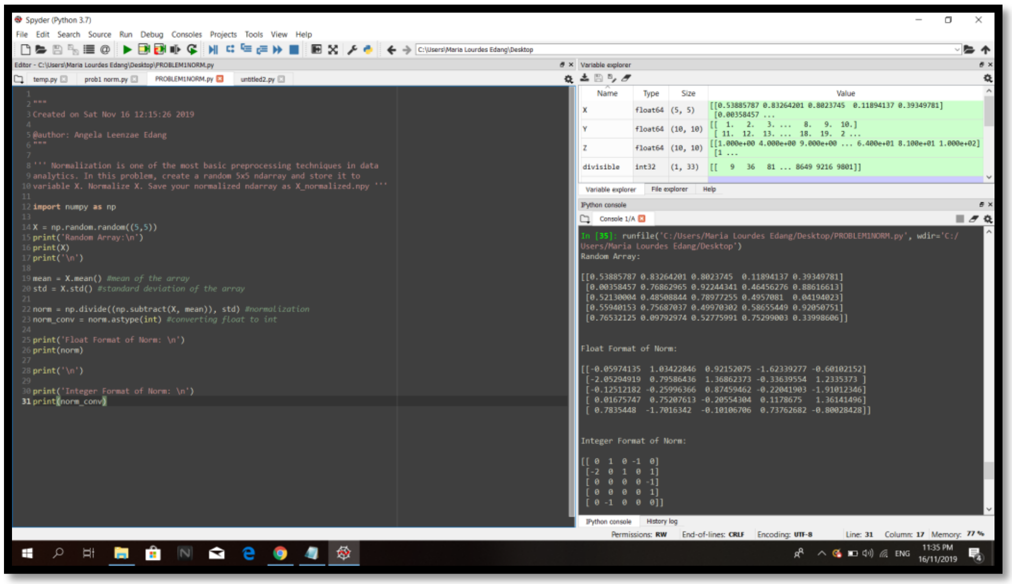
Lei-Ann Edang November 16, 2019

2ECE-A Programming Assignment

EXPERIMENT 7

NUMPY

Problem 1: *Normalization*

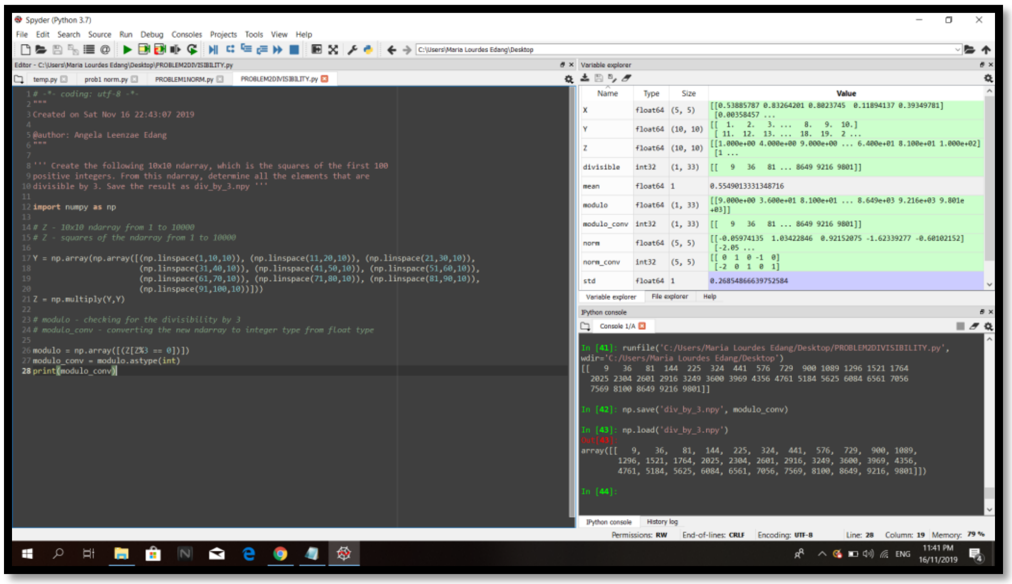


```python

"""
Created on Sat Nov 16 12:15:26 2019

@author: Angela Leenzae Edang
"""

''' Normalization is one of the most basic preprocessing techniques in data
analytics. In this problem, create a random 5x5 ndarray and store it to
variable X. Normalize X. Save your normalized ndarray as X_normalized.npy '''

import numpy as np

X = np.random.random((5,5))
print('Random Array:\n')
print(X)
print('\n')

mean = X.mean() #mean of the array
std = X.std() #standard deviation of the array

norm = np.divide((np.subtract(X, mean)), std) #normalization
norm_conv = norm.astype(int) #converting float to int

print('Float Format of Norm: \n')
print(norm)

print('\n')

print('Integer Format of Norm: \n')
print(norm_conv)
```

Name	Type	Size	Value
X	float64	(5, 5)	[[0.53885787 0.83264201 0.8023745 0.11894137 0.39349781] [0.00358457 ...
Y	float64	(10, 10)	[[1. 2. 3. ... 8. 9. 10.] [11. 12. 13. ... 18. 19. 2 ...
Z	float64	(10, 10)	[[1.000e+00 4.000e+00 9.000e+00 ... 6.400e+01 8.100e+01 1.000e+02] [1 ...
divisible	int32	(1, 33)	[[9 36 81 ... 8649 9216 9801]]

```
In [35]: runfile('C:/Users/Maria Lourdes Edang/Desktop/PROBLEM1NORM.py', wdir='C:/
Users/Maria Lourdes Edang/Desktop')
Random Array:

[[0.53885787 0.83264201 0.8023745  0.11894137 0.39349781]
 [0.00358457 0.76862965 0.92244341 0.46456276 0.88616613]
 [0.52130004 0.48508844 0.78977255 0.4957081  0.04194023]
 [0.55940153 0.75687037 0.49970302 0.58655449 0.92050751]
 [0.76532125 0.09792974 0.52775991 0.75299003 0.33998606]]


Float Format of Norm:

[[-0.05974135  1.03422846  0.92152075 -1.62339277 -0.60102152]
 [-2.05294919  0.79586436  1.36862373 -0.33639554  1.2335373 ]
 [-0.12512182 -0.25996366  0.87459462 -0.22041903 -1.91012346]
 [ 0.01675747  0.75207613 -0.20554304  0.1178675   1.36141496]
 [ 0.7835448  -1.7016342  -0.10106706  0.73762682 -0.80028428]]


Integer Format of Norm:

[[ 0  1  0 -1  0]
 [-2  0  1  0  1]
 [ 0  0  0  0 -1]
 [ 0  0  0  0  1]
 [ 0 -1  0  0  0]]
```

Problem 2: *Divisibility*



```python
# -*- coding: utf-8 -*-
"""
Created on Sat Nov 16 22:43:07 2019

@author: Angela Leenzae Edang
"""

''' Create the following 10x10 ndarray, which is the squares of the first 100
positive integers. From this ndarray, determine all the elements that are
divisible by 3. Save the result as div_by_3.npy '''

import numpy as np

# Z - 10x10 ndarray from 1 to 10000
# Z - squares of the ndarray from 1 to 10000

Y = np.array(np.array([(np.linspace(1,10,10)), (np.linspace(11,20,10)), (np.linspace(21,30,10)),
                       (np.linspace(31,40,10)), (np.linspace(41,50,10)), (np.linspace(51,60,10)),
                       (np.linspace(61,70,10)), (np.linspace(71,80,10)), (np.linspace(81,90,10)),
                       (np.linspace(91,100,10))]))
Z = np.multiply(Y,Y)

# modulo - checking for the divisibility by 3
# modulo_conv - converting the new ndarray to integer type from float type

modulo = np.array([(Z[Z%3 == 0])])
modulo_conv = modulo.astype(int)
print(modulo_conv)
```

Name	Type	Size	Value
X	float64	(5, 5)	[[0.53885787 0.83264201 0.8023745 0.11894137 0.39349781] [0.00358457 ...
Y	float64	(10, 10)	[[1. 2. 3. ... 8. 9. 10.] [11. 12. 13. ... 18. 19. 2 ...
Z	float64	(10, 10)	[[1.000e+00 4.000e+00 9.000e+00 ... 6.400e+01 8.100e+01 1.000e+02] [1 ...
divisible	int32	(1, 33)	[[9 36 81 ... 8649 9216 9801]]
mean	float64	1	0.5549013331348716
modulo	float64	(1, 33)	[[9.000e+00 3.600e+01 8.100e+01 ... 8.649e+03 9.216e+03 9.801e+03]]
modulo_conv	int32	(1, 33)	[[9 36 81 ... 8649 9216 9801]]
norm	float64	(5, 5)	[[-0.05974135 1.03422846 0.92152075 -1.62339277 -0.60102152] [-2.05 ...
norm_conv	int32	(5, 5)	[[0 1 0 -1 0] [-2 0 1 0 1]]
std	float64	1	0.26854866639752584

```
In [41]: runfile('C:/Users/Maria Lourdes Edang/Desktop/PROBLEM2DIVISIBILITY.py',
wdir='C:/Users/Maria Lourdes Edang/Desktop')
[[   9   36   81  144  225  324  441  576  729  900 1089 1296 1521 1764
  2025 2304 2601 2916 3249 3600 3969 4356 4761 5184 5625 6084 6561 7056
  7569 8100 8649 9216 9801]]

In [42]: np.save('div_by_3.npy', modulo_conv)

In [43]: np.load('div_by_3.npy')
Out[43]:
array([[   9,   36,   81,  144,  225,  324,  441,  576,  729,  900, 1089,
        1296, 1521, 1764, 2025, 2304, 2601, 2916, 3249, 3600, 3969, 4356,
        4761, 5184, 5625, 6084, 6561, 7056, 7569, 8100, 8649, 9216, 9801]])

In [44]:
```