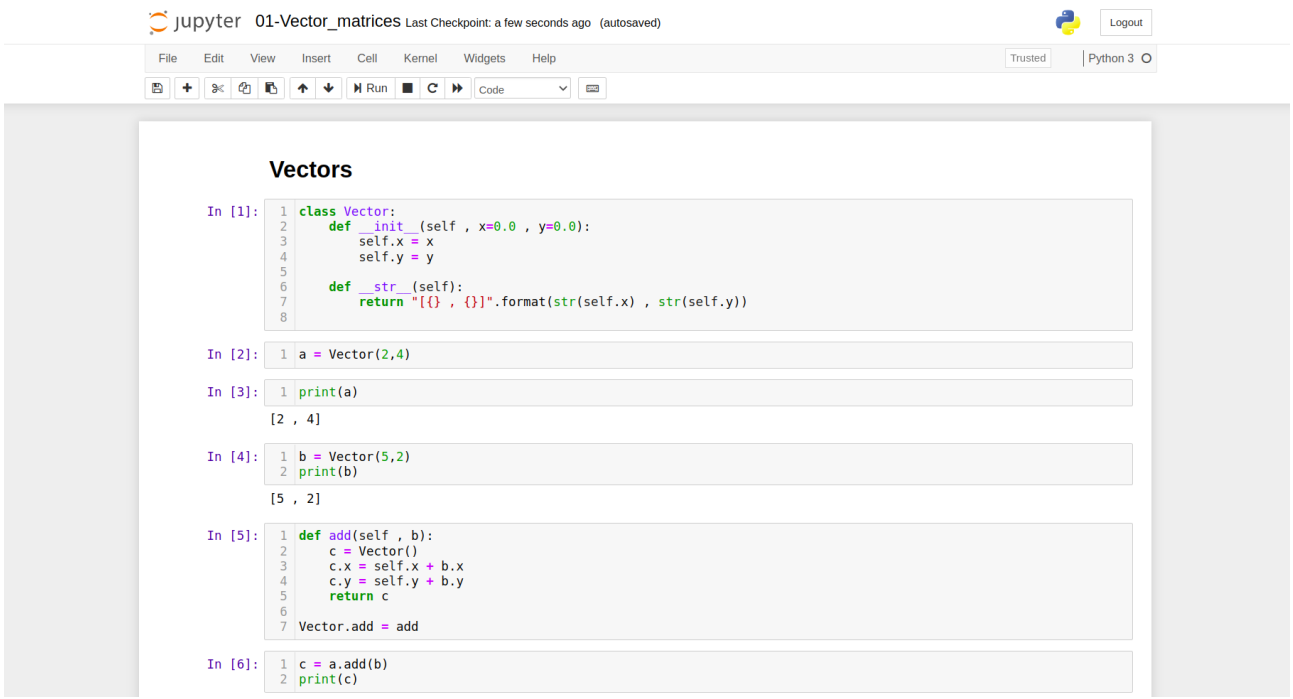
Assignment #2
Saad Ahmad
(20P-0051)
BS (CS) – 5A




```
In [6]:
c = a.add(b)
print(c)
```

```
[7 , 6]
```

```
In [7]:
def mul(self , s):
    return Vector(s * self.x , s * self.y)

Vector.mul = mul
```

```
In [8]:
d = a.mul(2)
print(d)
```

```
[4 , 8]
```

```
In [9]:
def sub(self , b):
    return self.add( b.mul(-1))

Vector.sub = sub
```

```
In [10]:
d_min_b = d.sub(b)
print(d_min_b)
```

```
[-1 , 6]
```

Matrix Representation (Dense)

In [11]:
```
class Matrix:

    def __init__(self , dims , fill):
        self.rows = dims[0]            # 3
        self.cols = dims[1]            # 4

        self.A = [
            [fill] * self.cols for i in range(self.rows)            #
        ]

```

In [12]:
```
m = Matrix((3,4) , 2.0)
```

In [13]:
```
print(m)
```

```
<__main__.Matrix object at 0x7ff81c625940>
```

In [14]:
```
def __str__(self):
    rows = len(self.A) # getting the first dimension
    ret = ''

    for i in range(rows):
        cols = len(self.A[i])
        for j in range(cols):
            ret += str(self.A[i][j]) + "\t"
        ret += "\n"
    return ret

Matrix.__str__ = __str__
```

In [15]:
```
print(m)
```

```
2.0     2.0     2.0     2.0
2.0     2.0     2.0     2.0
2.0     2.0     2.0     2.0
```

In [16]:
```
%time n = Matrix((100 , 100), 0.0)
```

```
CPU times: user 4.77 ms, sys: 185 µs, total: 4.96 ms
Wall time: 4.56 ms
```

Memory Usage

In [17]:
```
from sys import getsizeof
print(getsizeof(m))
print(getsizeof(n))
```

```
48
48
```

In [18]:
```
from pympler.asizeof import asizeof
```

In [19]:
```
asizeof(m) , asizeof(n)
```

Out[19]: (760, 86880)

In [20]:
```
dim = 5000
```

In [21]:
```
%time  m = Matrix((dim , dim) , 0.0)
```

```
CPU times: user 223 ms, sys: 117 ms, total: 340 ms
Wall time: 335 ms
```

In [22]:
```
size = asizeof(m)/(1024*1024)
print("{:.2f} MBs".format(size))
```

```
191.04 MBs
```

In [23]:
```
def get(self , i , j):
    if i<0 or i>self.rows:
        raise ValueError("Rows index out of range.")
    if j<0 or j>self.cols:
        raise ValueError("Column index out of range.")

    return self.A[i][j]
Matrix.get = get
```

In [24]:
```
m.get(1,2)
```

Out[24]:
```
0.0
```

In [25]:
```
m.get(15, 0)
```

Out[25]:
```
0.0
```

In [26]:
```
m.get(1,10)
```

Out[26]:
```
0.0
```

Matrix Representation (Sparse)

In [27]:
```
class Matrix:
    def __init__(self , dims):
        self.rows = dims[0]
        self.cols = dims[1]
        self.vals = {}
```

In [28]:
```
def set(self , i , j , val):
    self.vals[ (i,j) ] = val

Matrix.set = set
```

In [29]:
```
def get(self , i , j):
    if i<0 or i>self.rows:
        raise ValueError("Rows index out of range.")
    if j<0 or j>self.cols:
        raise ValueError("Column index out of range.")
    if (i,j) in self.vals:
        return self.vals[(i,j)]

    return 0.0

Matrix.get = get
```

In [30]:
```
m = Matrix((5,5))
```

In [31]:
```
print(m.vals)
```

```
{}
```

In [32]:
```
m.get(1,1)
```

Out[32]:
```
0.0
```

In [33]:
```
m.get(10,2)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-33-2353838e11b2> in <module>
----> 1 m.get(10,2)

<ipython-input-29-41dee7363588> in get(self, i, j)
      1 def get(self , i , j):
      2     if i<0 or i>self.rows:
----> 3         raise ValueError("Rows index out of range.")
      4     if j<0 or j>self.cols:
      5         raise ValueError("Column index out of range.")

ValueError: Rows index out of range.
```

In [34]:
```
m.set(1,2,15.0)
```

In [35]:
```
m.get(1,2)
```

Out[35]:
```
15.0
```

In [36]:
```
m.vals
```

Out[36]:
```
{(1, 2): 15.0}
```

In [37]:
```
m.set(1,4, 29.9)
```

In [38]:
```
m.get(1,4)
```

Out[38]:
```
29.9
```

In [39]:
```
dim = 500
m = Matrix((dim , dim))
```

In [40]:
```
asizeof(m)
```

Out[40]:
```
416
```

Numpy Intro

In [ ]:
```
%matplotlib inline
%run mplimp.py
```

In [2]:
```
import numpy as np
```

In [3]:
```
np.random.seed(1337)
```

Basics of Matrices

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

Out[4]:
```
array([1, 4, 3])
```

In [5]:
```
y = np.array([ [1, 4, 3],
               [9, 2, 7] ] )
y
```

Out[5]:
```
array([[1, 4, 3],
       [9, 2, 7]])
```

In [6]:
```
x.shape
```

Out[6]:
```
(3,)
```

In [7]:
```
y.shape
```

Out[7]:
```
(2, 3)
```

In [8]:
```
z = np.array(  [ [1, 4, 3] ]  )
```

In [9]:
```
z.shape
```

Out[9]:
```
(1, 3)
```

In [10]:
```
z = np.arange(1, 2000, 1)  # start, end, step
z[:10]
```

Out[10]:
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

In [11]:
```
z.shape
```

Out[11]:
```
(1999,)
```

In [12]:
```
np.arange(0.5, 3, 0.5)
```

Out[12]:
```
array([0.5, 1. , 1.5, 2. , 2.5])
```

In [13]:
```
np.arange(0.5, 10, 1).shape
```

Out[13]:
```
(10,)
```

In [14]:
```
np.arange(0.5, 10, 1).reshape(5, 2).shape
```

Out[14]:
```
(5, 2)
```

In [15]:
```
np.arange(0.5, 10, 1).reshape(5, 3).shape
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-15-7df1cdd41309> in <module>
----> 1 np.arange(0.5, 10, 1).reshape(5, 3).shape

ValueError: cannot reshape array of size 10 into shape (5,3)
```

In [16]:
```
# Evenly spaced but we don't know the step
np.linspace(3, 9, 10)
```

Out[16]:
```
array([3.        , 3.66666667, 4.33333333, 5.        , 5.66666667,
       6.33333333, 7.        , 7.66666667, 8.33333333, 9.        ])
```

In [17]:
```
print(x)
print(x[1])
print(x[1:])
```

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

In [18]:
```
print(y)
y[0, 1]
```

```
[[1 4 3]
 [9 2 7]]
```

Out[18]:
```
4
```

In [19]:
```
y[:, 1]
```

Out[19]:
```
array([4, 2])
```

In [20]:
```
y[:, [1, 2]]
```

Out[20]:
```
array([[4, 3],
       [2, 7]])
```

Matrix Operations

In [21]:
```
np.zeros((3, 5))
```

Out[21]:
```
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

In [22]:
```
np.ones((5, 3))
```

Out[22]:
```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

In [23]:
```
a = np.arange(1, 7)
a
```

Out[23]:
```
array([1, 2, 3, 4, 5, 6])
```

In [23]:
```
a = np.arange(1, 7)
a
```

Out[23]:
```
array([1, 2, 3, 4, 5, 6])
```

In [24]:
```
a.shape
```

Out[24]:
```
(6,)
```

In [25]:
```
a[3] = 7
a
```

Out[25]:
```
array([1, 2, 3, 7, 5, 6])
```

In [26]:
```
a[:3] = 1
a
```

Out[26]:
```
array([1, 1, 1, 7, 5, 6])
```

In [27]:
```
a[1:4] = [9, 8, 7]
a
```

Out[27]:
```
array([1, 9, 8, 7, 5, 6])
```

In [28]:
```
b = np.zeros((2, 2))
b[0, 0] = 1
b[0, 1] = 2
b[1, 1] = 4
b
```

Out[28]:
```
array([[1., 2.],
       [0., 4.]])
```

In [29]:
```
b.shape
```

Out[29]:
```
(2, 2)
```

Array Operations

In [30]:
```
print(b)
```

```
[[1. 2.]
 [0. 4.]]
```

In [31]:
```
b + 2
```

Out[31]:
```
array([[3., 4.],
       [2., 6.]])
```

In [32]:
```
2 * b
```

Out[32]:
```
array([[2., 4.],
       [0., 8.]])
```

In [33]:
```
b ** 2
```

Out[33]:
```
array([[ 1.,  4.],
       [ 0., 16.]])
```

In [34]:
```
sum(b)
```

Out[34]:
```
array([1., 6.])
```

In [34]:
```
sum(b)
```

Out[34]:
```
array([1., 6.])
```

In [35]:
```
b.sum()
```

Out[35]:
```
7.0
```

In [36]:
```
b
```

Out[36]:
```
array([[1., 2.],
       [0., 4.]])
```

In [37]:
```
b.sum(axis=0).shape
```

Out[37]:
```
(2,)
```

In [38]:
```
b.sum(axis=1).shape
```

Out[38]:
```
(2,)
```

In [39]:
```
b = np.array([[1, 2], [3, 4]])
d = np.array([[3, 4], [5, 6]])
```

In [40]:
```
print(b)
print(d)
```

```
[[1 2]
 [3 4]]
[[3 4]
 [5 6]]
```

In [41]:
```
b + d
```

Out[41]:
```
array([[ 4,  6],
       [ 8, 10]])
```

In [42]:
```
b * d
```

Out[42]:
```
array([[ 3,  8],
       [15, 24]])
```

In [43]:
```
b.dot(d)
```

Out[43]:
```
array([[13, 16],
       [29, 36]])
```

In [44]:
```
b ** d
```

Out[44]:
```
array([[   1,   16],
       [ 243, 4096]])
```

In [45]:
```
b.T
```

Out[45]:
```
array([[1, 3],
       [2, 4]])
```

In [46]:
```
a
```

Out[46]:
```
array([1, 9, 8, 7, 5, 6])
```

In [47]:
```
a.shape
```

Out[47]:
```
(6,)
```

In [48]:
```
a.T
```

Out[48]:
```
array([1, 9, 8, 7, 5, 6])
```

In [49]:
```
a.T.shape
```

Out[49]:
```
(6,)
```

In [50]:
```
a.reshape(6,1).T.shape
```

Out[50]:
```
(1, 6)
```

In [51]:
```
# Numpy has "broadcasting" or "mapping" functions
print(np.sqrt(36))

# works on both scalars and arrays
x = [1, 4, 9, 16]
np.sqrt(x)
```

```
6.0
```

Out[51]:
```
array([1., 2., 3., 4.])
```

In [52]:
```
x = np.array([1, 2, 4, 5, 9, 3])
y = np.array([0, 2, 3, 1, 2, 3])
```

In [53]:
```
x > 3
```

Out[53]:
```
array([False, False,  True,  True,  True, False])
```

In [54]:
```
x > y
```

Out[54]:
```
array([ True, False,  True,  True,  True, False])
```

Other Operations with Numpy

In [55]:
```
import math
def basic_sigmoid(x):
    """
    Compute sigmoid of x.

    Arguments:
    x -- A scalar

    Return:
    s -- sigmoid(x)
    """

    s = 1./(1. + math.e ** (-x))

    return s
```

In [56]:
```
basic_sigmoid(-1)
```

Out[56]:
```
0.2689414213699951
```

In [57]:
```
basic_sigmoid(0)
```

Out[57]:
```
0.5
```

In [58]:
```
x = [-1, 0, 3]
basic_sigmoid(x)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-58-f81e08c4b17c> in <module>
      1 x = [-1, 0, 3]
----> 2 basic_sigmoid(x)

<ipython-input-55-6b445ed2e437> in basic_sigmoid(x)
     11     """
     12
---> 13     s = 1./(1. + math.e ** (-x))
     14
     15     return s

TypeError: bad operand type for unary -: 'list'
```

In [59]:
```
import numpy as np

x = [-1, 0, 3]
x = np.array(x)
basic_sigmoid(x)
```

Out[59]:
```
array([0.26894142, 0.5       , 0.95257413])
```

Broadcasting

In [60]:
```
import numpy as np
```

In [61]:
```
# What is broadcasting?
x = np.array([1, 2, 3])
x * 3  # This makes sense
```

Out[61]:
```
array([3, 6, 9])
```

In [62]:
```
x + 3
```

Out[62]:
```
array([4, 5, 6])
```

In [63]:
```
x = np.arange(4)
xx = x.reshape(4, 1)
y = np.ones(5)
z = np.ones((3,4))

print("x =  ", x)
print("xx = ", xx)
print("y =  ", y)

print("Shapes: ")
print(x.shape)
print(xx.shape)
print(y.shape)
```

```
x =   [0 1 2 3]
xx =  [[0]
 [1]
 [2]
 [3]]
y =   [1. 1. 1. 1. 1.]
Shapes:
(4,)
(4, 1)
(5,)
```

In [64]:
```
x + y
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-64-cd60f97aa77f> in <module>
----> 1 x + y

ValueError: operands could not be broadcast together with shapes (4,) (5,)
```

In [65]:
```
xx.shape, x.shape
```

Out[65]:
```
((4, 1), (4,))
```

In [66]:
```
print(y)
print(xx)
```

```
[1. 1. 1. 1. 1.]
[[0]
 [1]
 [2]
 [3]]
```

In [67]:
```
out = xx + y
```

In [68]:
```
out
```

Out[68]:
```
array([[1., 1., 1., 1., 1.],
       [2., 2., 2., 2., 2.],
       [3., 3., 3., 3., 3.],
       [4., 4., 4., 4., 4.]])
```

In [69]:
```
out.shape
```

Out[69]:
```
(4, 5)
```

In [70]:
```
np.array([1]) + y
```

Out[70]:
```
array([2., 2., 2., 2., 2.])
```

In [71]:
```
print(z)
z.shape
```

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

Out[71]:
```
(3, 4)
```

In [72]:
```
x.shape
```

Out[72]:
```
(4,)
```

In [73]:
```
x
```

Out[73]:
```
array([0, 1, 2, 3])
```

In [74]:
```
z + x
```

Out[74]:
```
array([[1., 2., 3., 4.],
       [1., 2., 3., 4.],
       [1., 2., 3., 4.]])
```

In [75]:
```
a = np.array([[ 0.0 , 0.0,  0.0  ],
              [ 10.0, 10.0, 10.0 ],
              [ 20.0, 20.0, 20.0 ],
              [ 30.0, 30.0, 30.0 ]])

b = np.array( [ 1.0,  2.0,  3.0  ] )
```

In [76]:
```
a + b
```

Out[76]:
```
array([[ 1.,  2.,  3.],
       [11., 12., 13.],
       [21., 22., 23.],
       [31., 32., 33.]])
```

In [77]:
```
a = np.array([[ 0.0 , 0.0,  0.0  , 0.0  ],
              [ 10.0, 10.0, 10.0 , 10.0 ],
              [ 20.0, 20.0, 20.0 , 20.0 ],
              [ 30.0, 30.0, 30.0 , 30.0 ]])

b = np.array( [ 1.0,  2.0] )
```

In [78]:
```
a + b
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-78-bd58363a63fc> in <module>
----> 1 a + b

ValueError: operands could not be broadcast together with shapes (4,4) (2,)
```

In [79]:
```
a = np.array( [ 0.0, 10.0, 20.0, 30.0 ] )
b = np.array( [ 1.0, 2.0,  3.0 ]        )
```

In [80]:
```
a + b
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-80-bd58363a63fc> in <module>
----> 1 a + b

ValueError: operands could not be broadcast together with shapes (4,) (3,)
```

In [81]:
```
print(a.shape, b.shape)
```

```
(4,) (3,)
```

In [82]:
```
a1 = a.reshape(4, 1)
```

In [83]:
```
a1 + b
```

Out[83]:
```
array([[ 1.,  2.,  3.],
       [11., 12., 13.],
       [21., 22., 23.],
       [31., 32., 33.]])
```

In [84]:
```
print(a1.shape, b.shape)
```

```
(4, 1) (3,)
```

In [85]:
```
print(a1)
```

```
[[ 0.]
 [10.]
 [20.]
 [30.]]
```

Normalizing Rows and Columns

In [86]:
```
x = np.array([ [ 0, 3, 4 ],
               [ 1, 6, 4 ] ])
```

In [87]:
```
x_norm = np.amax(x, axis=1)   # get row-wise max
print(x_norm)
```

```
[4 6]
```

In [88]:
```
x / x_norm
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-88-8a1a7e8482dd> in <module>
----> 1 x / x_norm

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

In [89]:
```
print(x.shape)
print(x_norm.shape)
```

```
(2, 3)
(2,)
```

In [90]:
```
x_norm = x_norm.reshape(2, 1)
```

In [91]:
```
x_norm
```

Out[91]:
```
array([[4],
       [6]])
```

In [92]:
```
x / x_norm
```

Out[92]:
```
array([[0.        , 0.75      , 1.        ],
       [0.16666667, 1.        , 0.66666667]])
```

In [93]:
```
# We can also use another normalization method
x = np.array([ [ 0, 3, 4 ],
               [ 1, 6, 4 ] ])

# no need to reshape again
x_norm = np.linalg.norm(x, ord = 2, axis = 1, keepdims = True)
x / x_norm
```

Out[93]:
```
array([[0.        , 0.6       , 0.8       ],
       [0.13736056, 0.82416338, 0.54944226]])
```

In [94]:
```
x_norm.shape
```

Out[94]:
```
(2, 1)
```

Reshaping Revisited

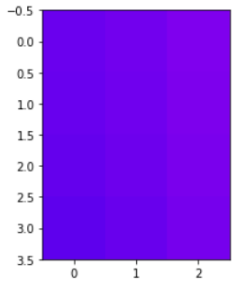
In [95]:
```
i = np.array([
    [  [ 0.1,  0.1,  0.9 ], [ 0.2,  0.1,  0.9 ], [ 0.3,  0.1,  0.9 ]  ],

    [  [ 0.1,  0.2,  0.9 ], [ 0.2,  0.2,  0.9 ], [ 0.3,  0.2,  0.9 ]  ],

    [  [ 0.1,  0.3,  0.9 ], [ 0.2,  0.3,  0.9 ], [ 0.3,  0.3,  0.9 ]  ],

    [  [ 0.1,  0.4,  0.9 ], [ 0.2,  0.4,  0.9 ], [ 0.3,  0.4,  0.9 ]  ]

])
```

In [96]:
```
i.shape
```

Out[96]:
```
(4, 3, 3)
```

In [97]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
```

In [98]:
```
1 _ = plt.imshow(i)
```



In [99]:
```
1 i.reshape(36)
```

Out[99]:
```
array([0.1, 0.1, 0.9, 0.2, 0.1, 0.9, 0.3, 0.1, 0.9, 0.1, 0.2, 0.9, 0.2,
       0.2, 0.9, 0.3, 0.2, 0.9, 0.1, 0.3, 0.9, 0.2, 0.3, 0.9, 0.3, 0.3,
       0.9, 0.1, 0.4, 0.9, 0.2, 0.4, 0.9, 0.3, 0.4, 0.9])
```

In [100]:
```
1 print(i.shape)
2 i_sh = i.shape
3 i.reshape(i_sh[0] * i_sh[1] * i_sh[2], 1)
```

```
(4, 3, 3)
```

(4, 3, 3)

Out[100]:
```
array([[0.1],
       [0.1],
       [0.9],
       [0.2],
       [0.1],
       [0.9],
       [0.3],
       [0.1],
       [0.9],
       [0.1],
       [0.2],
       [0.9],
       [0.2],
       [0.2],
       [0.9],
       [0.3],
       [0.2],
       [0.9],
       [0.1],
       [0.3],
       [0.9],
       [0.2],
       [0.3],
       [0.9],
       [0.3],
       [0.3],
       [0.9],
       [0.1],
       [0.4],
       [0.9],
       [0.2],
       [0.4],
       [0.9],
       [0.3],
       [0.4],
       [0.9]])
```

Vectorization

In [101]:
```
dim = 100
```

In [102]:
```
A = np.random.rand(dim, dim)
B = np.random.rand(dim, dim)
```

In [103]:
```
A[0].size, A[1].size
```

Out[103]:
```
(100, 100)
```

In [104]:
```
def add_arrays(A, B):
    C = np.zeros((A[0].size, A[1].size))

    for i in range(A[0].size):
        for j in range(A[1].size):
            C[i, j] = A[i, j] + B[i, j]
    return C
```

In [105]:
```
%time C = add_arrays(A, B)
```

```
CPU times: user 5.38 ms, sys: 262 µs, total: 5.64 ms
Wall time: 5.53 ms
```

In [106]:
```
%time C = A + B
```

```
CPU times: user 171 µs, sys: 35 µs, total: 206 µs
Wall time: 155 µs
```

In [107]:
```
import time

# Non-vectorized time
start = time.time()

C = add_arrays(A, B)

end = time.time()
non_vec_time = end - start


## Vectorized time
start = time.time()

C = A + B

end = time.time()
vec_time = end - start
```

In [108]:
```
vec_time / non_vec_time * 100
```

Out[108]:
```
2.956498207157838
```

In [109]:
```
%%time
total = 0
for i in np.arange(100_000_000):
    total += i
print(total)
```

```
4999999950000000
CPU times: user 14.6 s, sys: 79.3 ms, total: 14.7 s
Wall time: 14.7 s
```

In [110]:
```
%time sum(np.arange(100_000_000))
```

```
CPU times: user 7.4 s, sys: 100 ms, total: 7.5 s
Wall time: 7.47 s
```

Out[110]:
```
4999999950000000
```

In [ ]:
```

```