```
#!/usr/bin/env python
# coding: utf-8

# In[ ]:





# In[1]:

#Instructions to run
#This python file is extracted from the jupyter notebook version
#Before running the program make sure CUDA, Python and Numba are installed correctly
#To run the program, type: python Numba_example.py
#Source code reference:
https://numba.pydata.org/numba-doc/latest/cuda/examples.html#matrix-multiplication
#Source code reference: https://nyu-cds.github.io/python-numba/

from numba import cuda, float32
import numpy
import numba
import math

# Controls threads per block and shared memory usage.
# The computation will be done on blocks of TPBxTPB elements.
TPB = 16

@cuda.jit
def kernel_op(A, B, C):

    #cuda.grid returns the absolute position of the current thread in the entire grid of blocks
    x, y = cuda.grid(2)

    if x >= C.shape[0] and y >= C.shape[1]:
        # Quit if (x, y) is outside of valid C boundary
        return

    # Each thread computes one element in the result matrix.
    C[x,y]=A[x,y]-B[x,y]


# In[2]:
```

```
# Initialite the data array
A = numpy.ones([48,48], dtype = float)
B = numpy.ones([48,48], dtype =float)

#copy the host variables to device
A_global_mem = cuda.to_device(A)
B_global_mem = cuda.to_device(B)

#Create memory for C in device
C_global_mem = cuda.device_array((48,48))

# Configure the blocks
threadsperblock = (TPB, TPB)
blockspergrid_x = int(math.ceil(A.shape[0] / threadsperblock[1]))
blockspergrid_y = int(math.ceil(B.shape[1] / threadsperblock[0]))
blockspergrid = (blockspergrid_x, blockspergrid_y)


# In[3]:


# Start the kernel
kernel_op[blockspergrid, threadsperblock](A_global_mem, B_global_mem, C_global_mem)
#copy the result to CPU
res = C_global_mem.copy_to_host()


# In[4]:


print(res)


# In[5]:


print("The sum is ",numpy.sum(res))


# In[ ]:
```