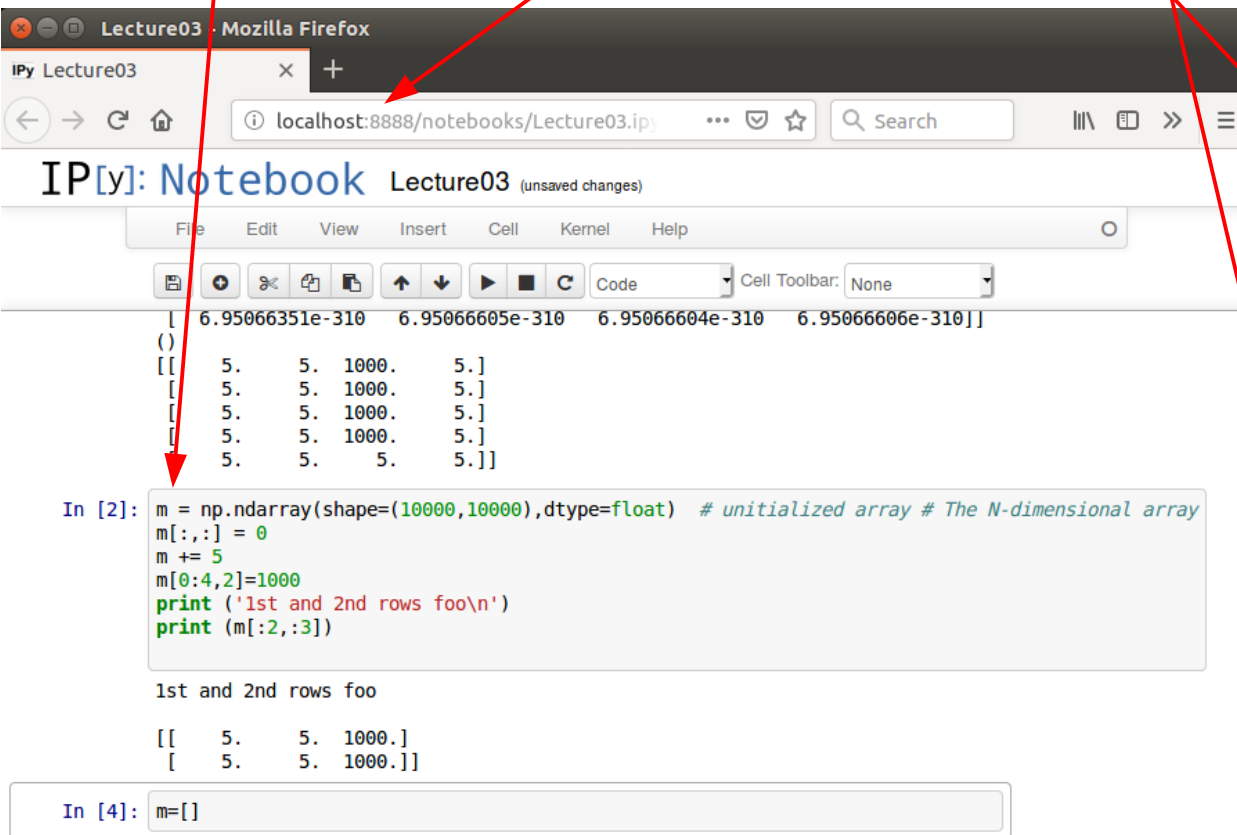


Physics 77  
Fall 2019  
Week 5

These slides summarize the Lecture  
discussion of Memory Allocation

Here we are running the Python notebook on the local machine, so we can monitor details about CPU usage and Memory usage. This cell causes about 800 MB of memory to be allocated. (The square of 10000 times 8 bytes). After the cell is evaluated, the memory is still in use.



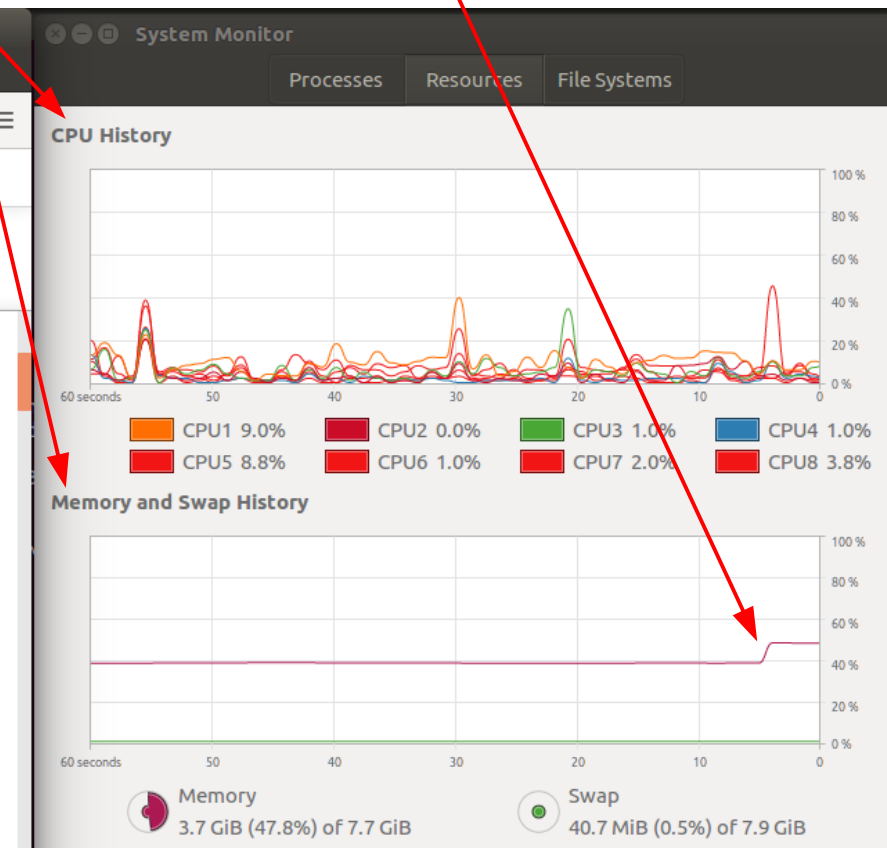
```
[ 6.95066351e-310  6.95066605e-310  6.95066604e-310  6.95066606e-310]]
()
[[ 5.  5. 1000.  5.]
 [ 5.  5. 1000.  5.]
 [ 5.  5. 1000.  5.]
 [ 5.  5. 1000.  5.]
 [ 5.  5.  5.  5.]]

In [2]: m = np.ndarray(shape=(10000,10000),dtype=float) # uninitialized array # The N-dimensional array
m[:, :] = 0
m += 5
m[0:4,2]=1000
print ('1st and 2nd rows foo\n')
print (m[:2,:3])

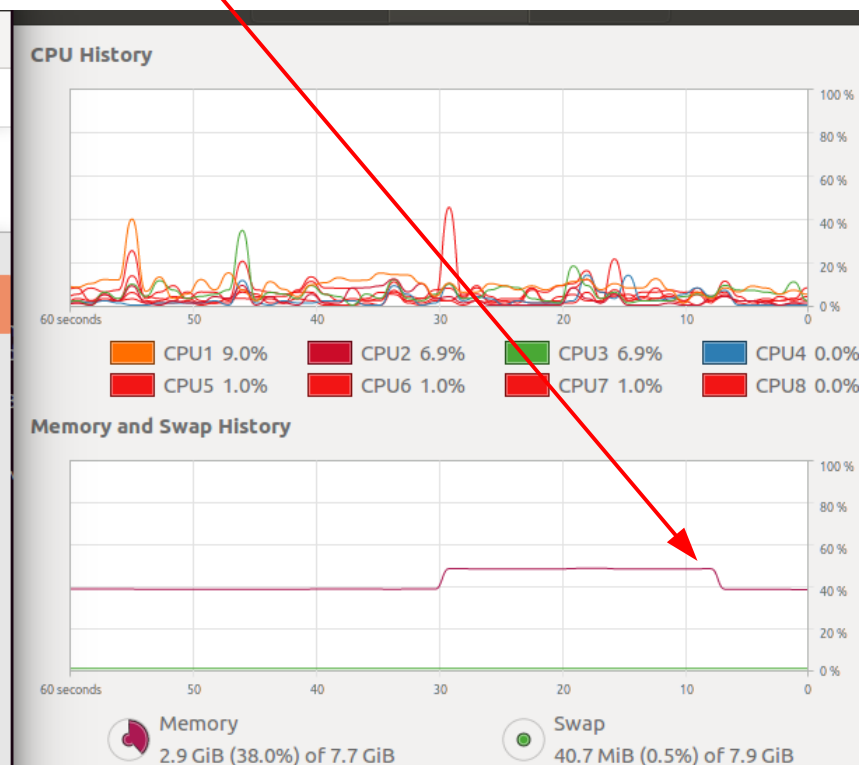
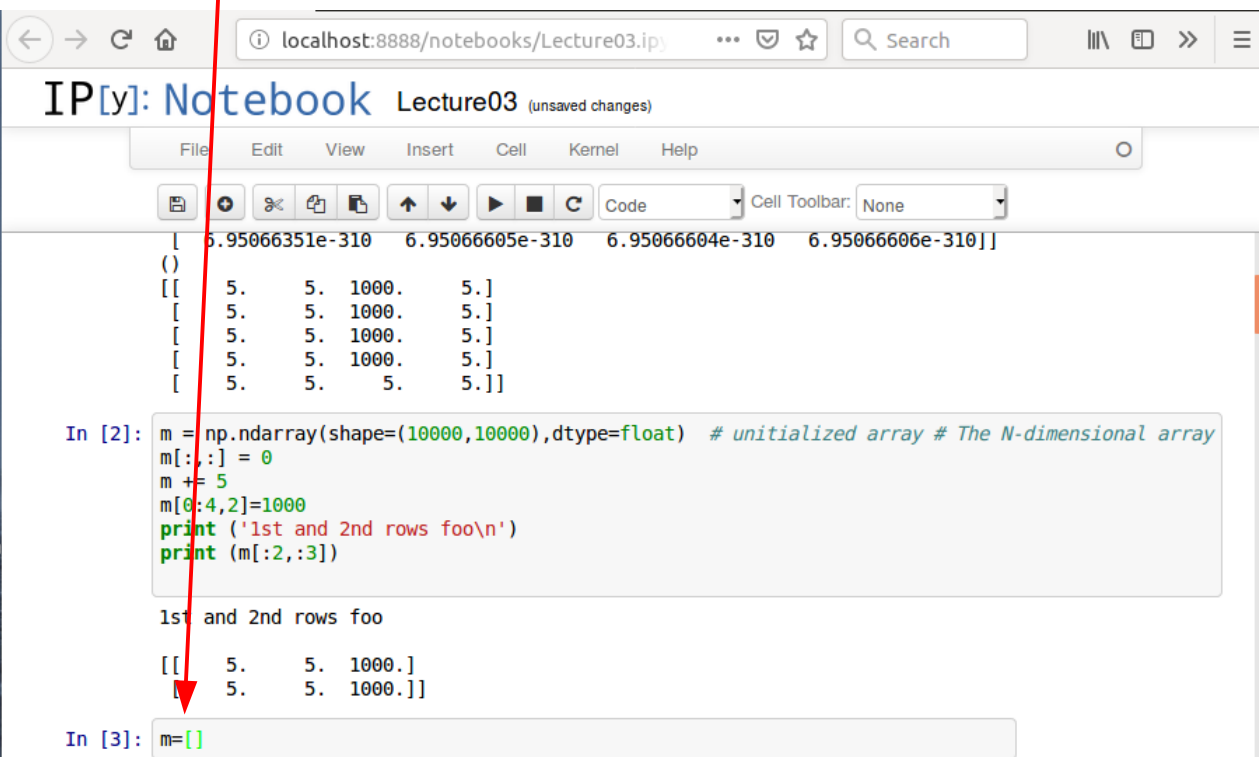
1st and 2nd rows foo

[[ 5.  5. 1000.]
 [ 5.  5. 1000.]]

In [4]: m=[]
```



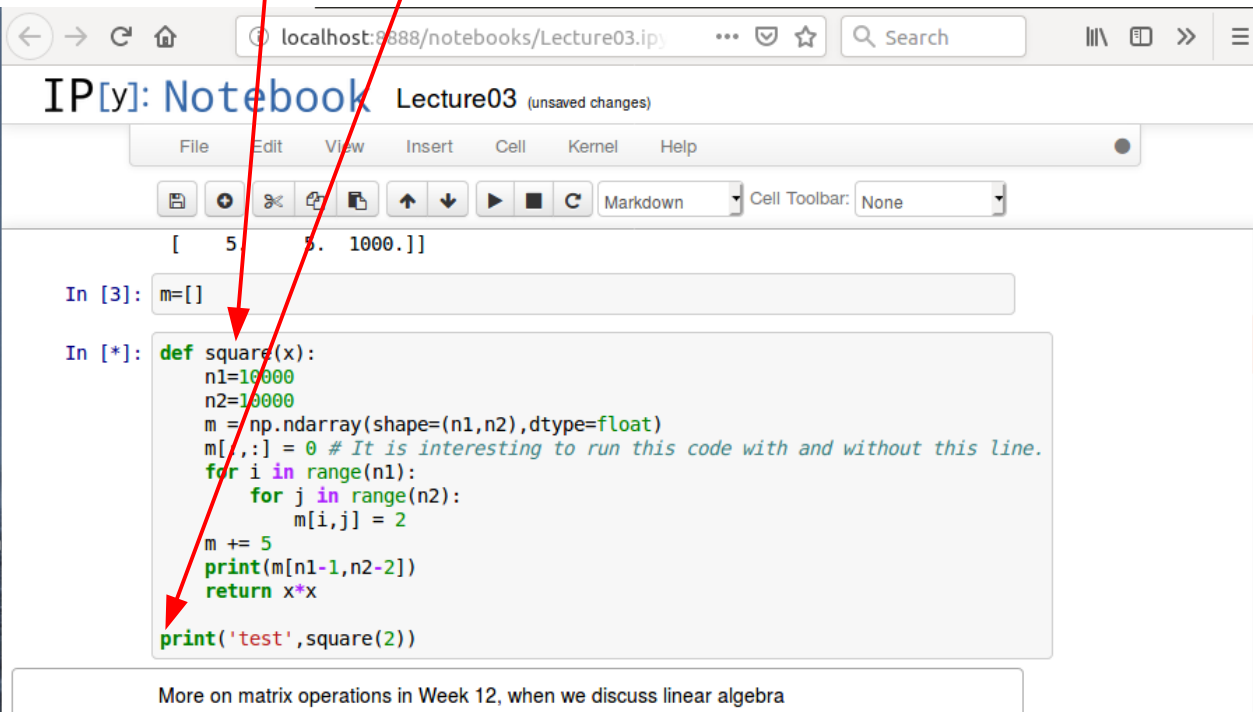
This cell causes the memory to be freed. There is very little CPU usage thus far.



This function allocates about 800 MB of memory internally and does some calculations that take about 20 seconds. The screen shot was taken about 8 seconds after we called the function.

We see the CPU usage go to 100% for one of the CPUs.

We see the memory allocated.



IP[y]: Notebook Lecture03 (unsaved changes)

File Edit View Insert Cell Kernel Help

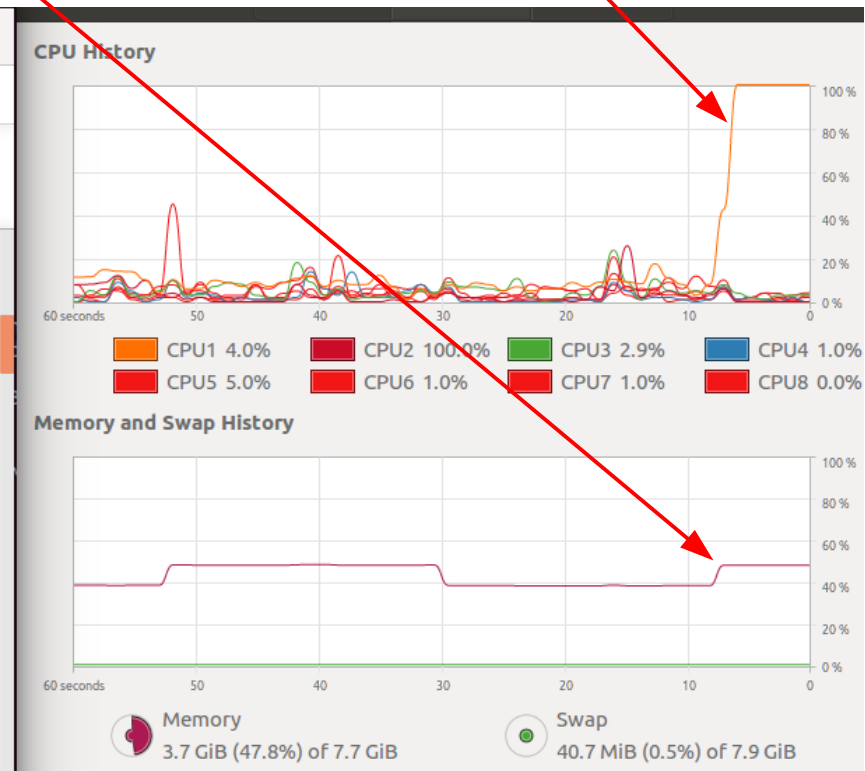
Markdown Cell Toolbar: None

```
[ 5      8. 1000.]]
```

```
In [3]: m=[]
```

```
In [*]: def square(x):
        n1=10000
        n2=10000
        m = np.ndarray(shape=(n1,n2),dtype=float)
        m[:,:] = 0 # It is interesting to run this code with and without this line.
        for i in range(n1):
            for j in range(n2):
                m[i,j] = 2
        m += 5
        print(m[n1-1,n2-2])
        return x*x
        print('test',square(2))
```

More on matrix operations in Week 12, when we discuss linear algebra



The function has returned, as evidenced by the fact that we see the text output printed. The memory is automatically freed because its allocation was internal to the function. The CPU usage was the equivalent of 100% utilization of one CPU, as expected for the single-thread calculation. For reasons that apply to other calculations, not just Python, the calculation was handed off from one CPU to another about half way through; the sum of the utilizations was 100% throughout the hand off.

## IP[y]: Notebook Lecture03 (unsaved changes)

File Edit View Insert Cell Kernel Help

File Edit View Insert Cell Kernel Help

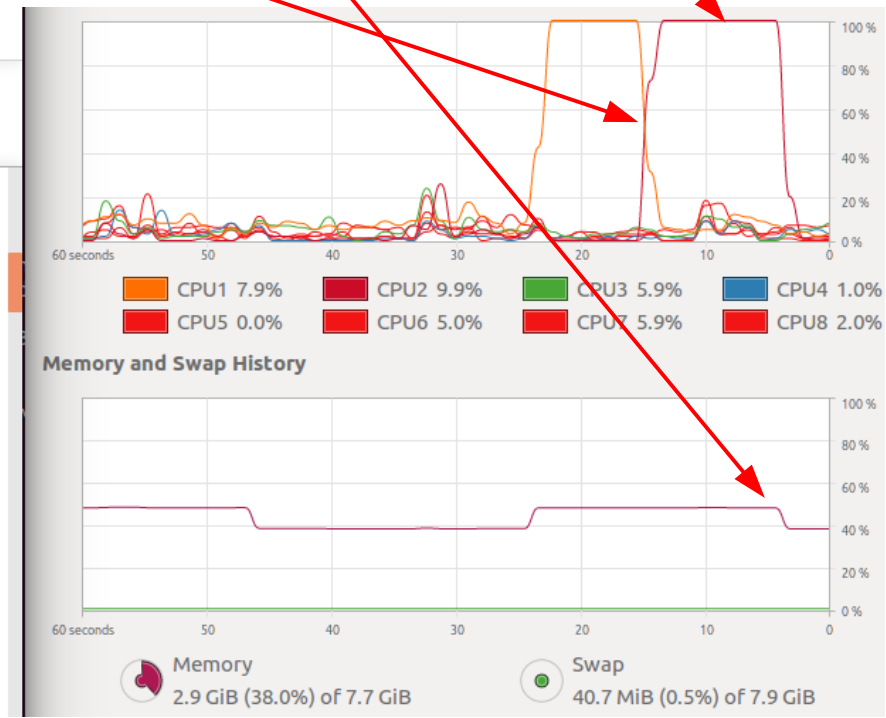
[ 5. 5. 1000.]

In [3]: m=[]

```
In [4]: def square(x):
n1=10000
n2=10000
m = np.ndarray(shape=(n1,n2),dtype=float)
m[:,:] = 0 # It is interesting to run this code with and without this line.
for i in range(n1):
    for j in range(n2):
        m[i,j] = 2
    m += 5
    print(m[n1-1,n2-2])
    return x*x
print('test',square(2))
```

7.0  
('test', 4)

More on matrix operations in Week 12, when we discuss linear algebra



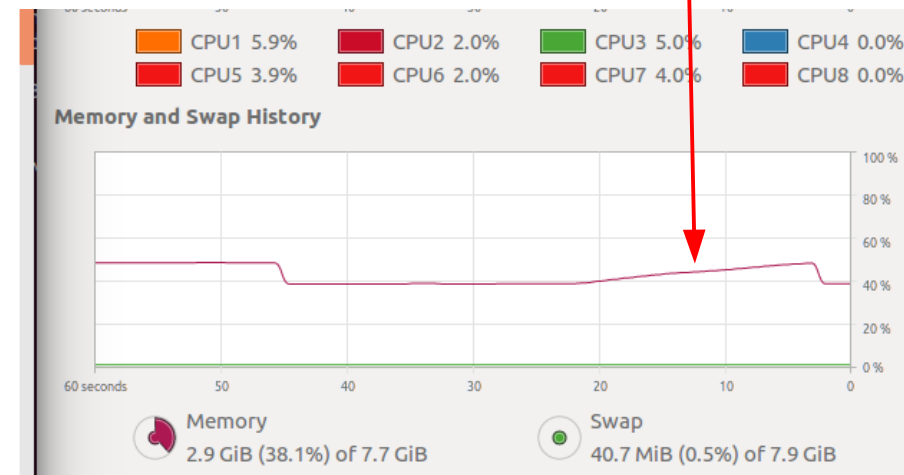
Here we have commented out the line that zeros out the memory. The result is that memory is gradually allocated over the course of the 20 second calculation, as we start to use it. This can be seen by the gradual ramp-up in memory usage.

```
In [5]: def square(x):
n1=10000
n2=10000
m = np.ndarray(shape=(n1,n2),dtype=float)
#m[:, :] = 0 # It is interesting to run this code with and without this line.
for i in range(n1):
    for j in range(n2):
        m[i,j] = 2
    m += 5
    print(m[n1-1,n2-2])
    return x*x

print('test',square(2))

7.0
('test', 4)
```

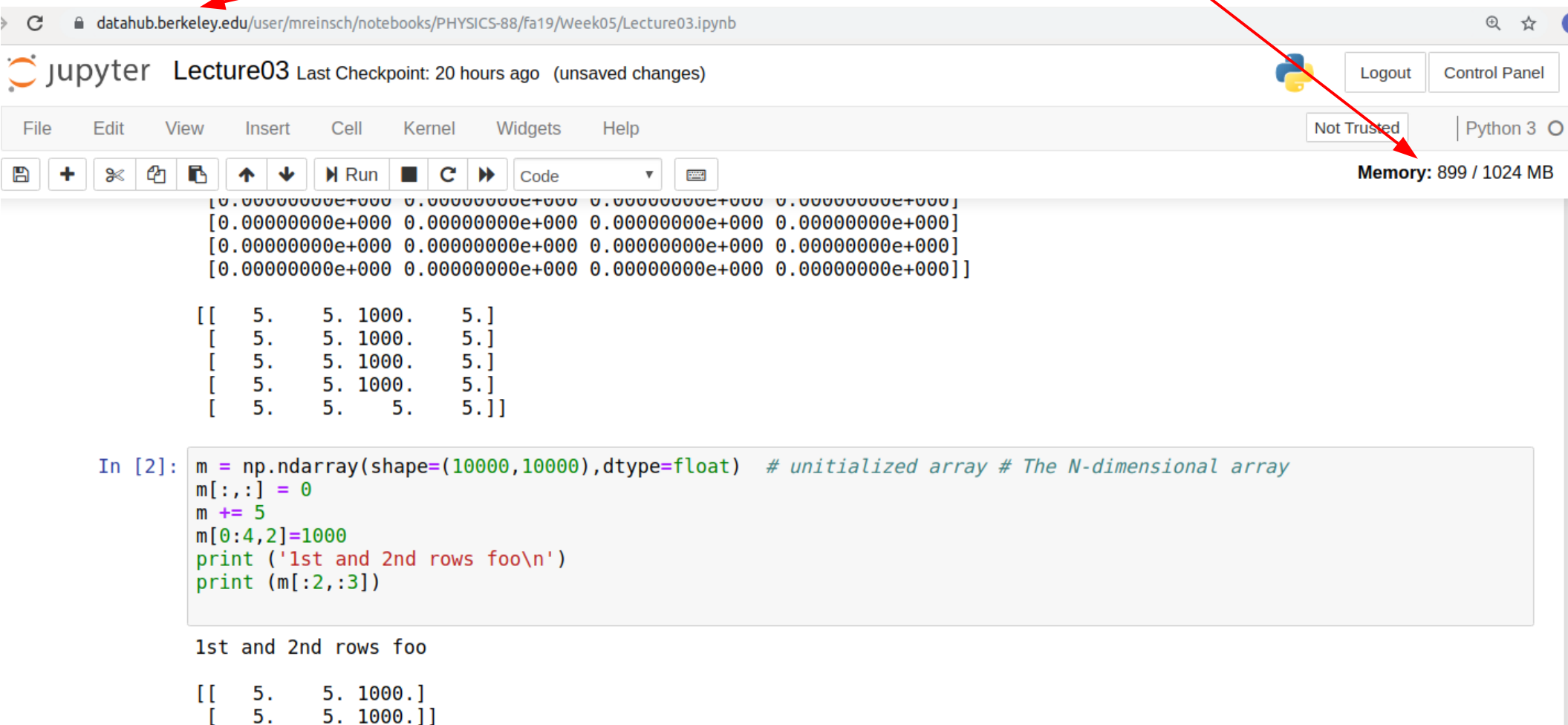
More on matrix operations in Week 12, when we discuss linear algebra



The previous slides were for the case of a notebook running on the local machine.

Next, let's repeat the analysis for a notebook running on the datahub.

Here we have evaluated the first cell that caused 800 MB to be allocated. Our Memory usage went up by about 800 MB on this indicator.



datahub.berkeley.edu/user/mreinsch/notebooks/PHYSICS-88/fa19/Week05/Lecture03.ipynb

jupyter Lecture03 Last Checkpoint: 20 hours ago (unsaved changes)

Logout Control Panel

Not Trusted Python 3

Memory: 899 / 1024 MB

```
[0.00000000e+000 0.00000000e+000 0.00000000e+000 0.00000000e+000]
[0.00000000e+000 0.00000000e+000 0.00000000e+000 0.00000000e+000]
[0.00000000e+000 0.00000000e+000 0.00000000e+000 0.00000000e+000]
[0.00000000e+000 0.00000000e+000 0.00000000e+000 0.00000000e+000]]

[[ 5.    5. 1000.    5.]
 [ 5.    5. 1000.    5.]
 [ 5.    5. 1000.    5.]
 [ 5.    5. 1000.    5.]
 [ 5.    5.    5.    5.]]
```

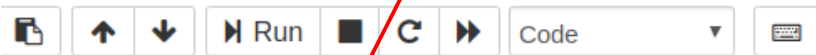
In [2]:

```
m = np.ndarray(shape=(10000,10000),dtype=float) # uninitialized array # The N-dimensional array
m[:, :] = 0
m += 5
m[0:4,2]=1000
print ('1st and 2nd rows foo\n')
print (m[:2,:3])
```

1st and 2nd rows foo

```
[[ 5.    5. 1000.]
 [ 5.    5. 1000.]]
```

After we evaluate this cell, the memory usage returns to its previous value, indicating that the approx 800 MB has been freed.



Memory: 136 / 1024 MB

```
m[:, :] = 0
m += 5
m[0:4, 2] = 1000
print ('1st and 2nd rows foo\n')
print (m[:2, :3])
```

1st and 2nd rows foo

```
[[ 5.  5. 1000.]
 [ 5.  5. 1000.]
```

```
m=[]
```



We have evaluated this cell, and the calculation is running, as evidenced by these symbols (asterisk and hourglass).

The memory indicator has not yet registered the increase. It is updated every so often. A few seconds later it was showing this larger value.

Memory: 899 / 1024 MB

The screenshot shows a Jupyter Notebook interface in a web browser. The browser tabs include "s.com - Leag...", "Demonstrations By UCB...", "PHYSICS-88/fa19/Week05...", "Inbox - mreinsch@berkele...", "Introduction to Computa...", and "Lecture03". The address bar shows "datahub.berkeley.edu/user/mreinsch/notebooks/PHYSICS-88/fa19/Week05/Lecture03.ipynb". The Jupyter interface has a top bar with "Lecture03", "Last Checkpoint: 20 hours ago (unsaved changes)", a Python logo, "Logout", and "Control Panel". Below this is a menu bar with "Edit", "View", "Insert", "Cell", "Kernel", "Widgets", and "Help". A toolbar contains icons for undo, redo, copy, paste, run, and other actions. The main area shows a code cell with the following code:

```
In [3]: m=[]
```

```
In [*]: def square(x):
    n1=10000
    n2=10000
    m = np.ndarray(shape=(n1,n2),dtype=float)
    m[:,:] = 0 # It is interesting to run this code with and without this line.
    for i in range(n1):
        for j in range(n2):
            m[i,j] = 2
    m += 5
    print(m[n1-1,n2-2])
    return x*x

print('test',square(2))
```

At the bottom right of the interface, the memory usage is displayed as "Memory: 173 / 1024 MB".

A Linux System Monitor was used for some of the previous slides. If you are running Windows, you can do the same thing with the Task Manager, shown below. If you have a Mac, there are similar utilities you can use.

