UT Southwestern Medical Center
Lyda Hill Department of Bioinformatics | BioHPC

python™ on BioHPC

[web] portal.biohpc.swmed.edu
[email] biohpc-help@utsouthwestern.edu

Jan 13, 2021

Overview

- Running Python on BioHPC
- Conda environment
- Jupyter Notebook and JupyterLab on Demand
- Profiling - measure the Python script execution time
- Popular python packages - Numpy, Scipy and Matplotlib
- Brief introduction on multiprocessing and MPI

Why Python?

1

Beginner Friendly

A clean, simple, readable, and easy to learn programming language

2

Flexible, extensible & Versatile

Python is portable and applicable in all environments—”Python is the glue”

3

Community & Libraries

An abundant source of community created libraries and frameworks

4

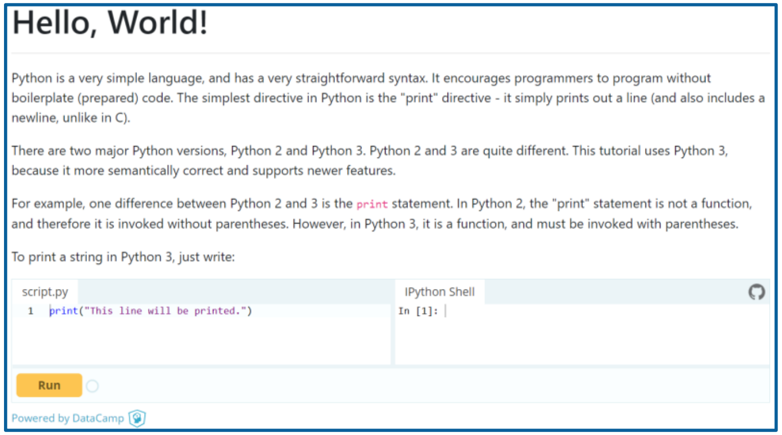
Popularity for Scientific Computing

Big data handling, analysis, and visualization, machine learning, artificial intelligence

The Scientific Python Stack

jupyter
NumPy
SciPy
CONDA
pandas
matplotlib
scikit
learn

Learning Python from Scratch



Hello, World!

Python is a very simple language, and has a very straightforward syntax. It encourages programmers to program without boilerplate (prepared) code. The simplest directive in Python is the "print" directive - it simply prints out a line (and also includes a newline, unlike in C).

There are two major Python versions, Python 2 and Python 3. Python 2 and 3 are quite different. This tutorial uses Python 3, because it more semantically correct and supports newer features.

For example, one difference between Python 2 and 3 is the `print` statement. In Python 2, the "print" statement is not a function, and therefore it is invoked without parentheses. However, in Python 3, it is a function, and must be invoked with parentheses.

To print a string in Python 3, just write:

```
print("This line will be printed.")
```

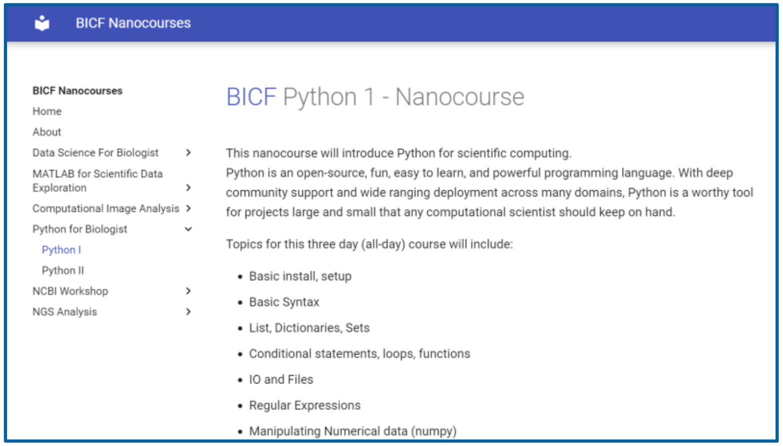
```
In [1]:
```

https://www.learnpython.org/

Free, Interactive Python Tutorial

No sign up needed

Great for new programmers



BICF Python 1 - Nanocourse

This nanocourse will introduce Python for scientific computing.
Python is an open-source, fun, easy to learn, and powerful programming language. With deep community support and wide ranging deployment across many domains, Python is a worthy tool for projects large and small that any computational scientist should keep on hand.

Topics for this three day (all-day) course will include:

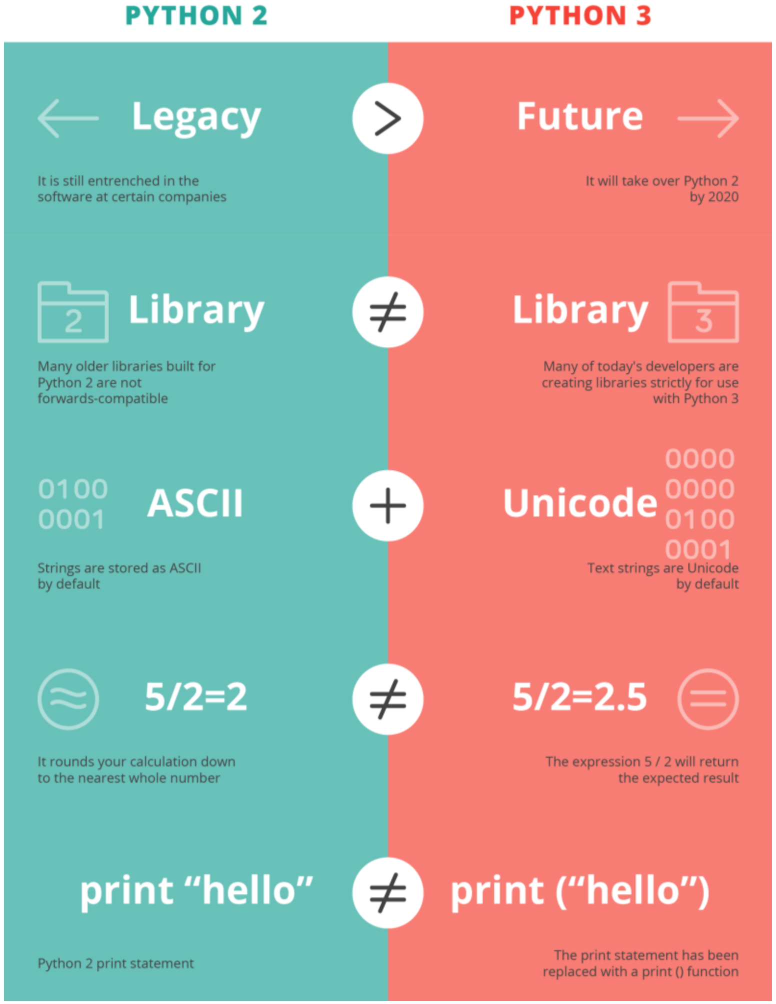
- Basic install, setup
- Basic Syntax
- List, Dictionaries, Sets
- Conditional statements, loops, functions
- IO and Files
- Regular Expressions
- Manipulating Numerical data (numpy)

https://bicf.pages.biohpc.swmed.edu/bicf_nanocourses/python_1/

Python Nanocourse for Graduate students and Postdocs

Registration is needed once available (Currently not available)

Python 2 vs Python 3



PYTHON 2

PYTHON 3

Legacy > Future

It is still entrenched in the software at certain companies

It will take over Python 2 by 2020

Library ≠ Library

Many older libraries built for Python 2 are not forwards-compatible

Many of today's developers are creating libraries strictly for use with Python 3

ASCII + Unicode

Strings are stored as ASCII by default

Text strings are Unicode by default

5/2=2 ≠ 5/2=2.5

It rounds your calculation down to the nearest whole number

The expression 5 / 2 will return the expected result

print "hello" ≠ print ("hello")

Python 2 print statement

The print statement has been replaced with a print () function

[Python 2 vs Python 3: Which Should I Learn?]

- Python 3.x made **backward incompatible** changes.
- Python 2 support officially stopped January 1 2020.
- Python 3 is recommended for new development.

Run Python on BioHPC

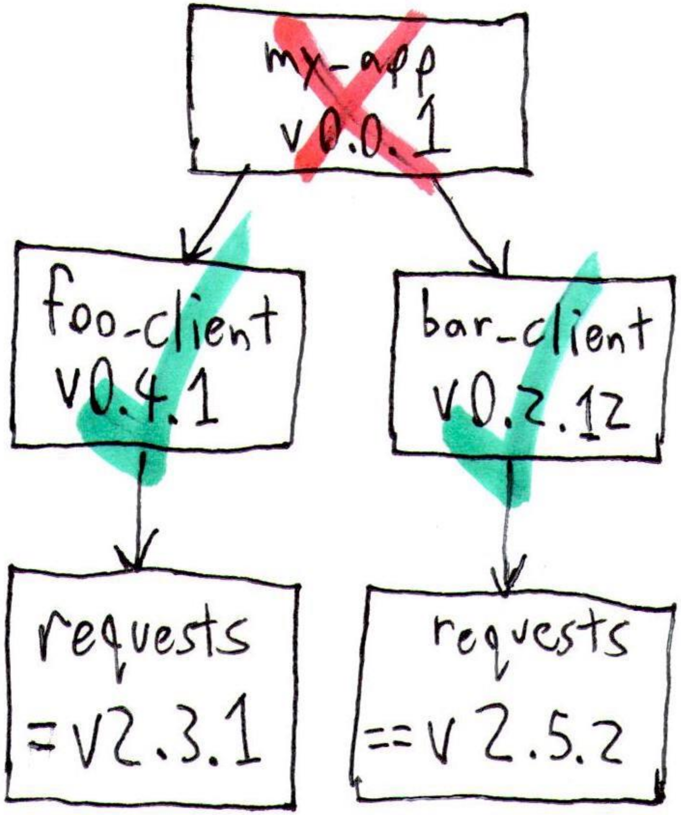
- Python 2.7 by default comes with RHEL 7 on BioHPC nodes

```
[s123456@Nucleus006 ~]$ python
Python 2.7.5 (default, Jun 11 2019, 14:33:56)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-39)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- More Python versions can be load from $ module

```
[s123456@Nucleus006 ~]$ module avail python

--------------------------- /cm/shared/modulefiles ---------------------------
python/2.7.14-anaconda          python/3.4.x-anaconda
python/2.7.3-epd                python/3.6.1-2-anaconda
python/2.7.5                    python/3.6.4-anaconda
python/2.7.6-epd                python/3.7.x-anaconda
python/2.7.x-anaconda
python/3.3.2
```

Challenge in Python



Dependency Hell

Affects all modern languages, especially interpreted ones.

Python especially challenging:

- Huge number of 3rd party packages
- Rapidly changing APIs
- Scientific packages need non-python dependencies.

Solutions - Conda / virtualenv etc…

Conda environment

- Dependencies
- Conda is a package manager, also serve as environment manager, allows you to have multiple isolated environment for different projects on a single machine
  - Project A: Python 2.7 and Biopython 1.60
  - Project B: Python 3.5 and Biopython 1.68
- Anaconda distribution: popular python/R data science platform, a collection of 7500+ packages
- The newly created environment will be installed in the directory

```
/home2/<username>/.conda/envs
```

- Ref to https://portal.biohpc.swmed.edu/content/guides/conda-biohpc

Anaconda – Default Environment

```
$ module load python/3.7.x-anaconda
```

291 packages, including full scientific python stack

```
$ conda list
```

Web Visualization – for software need GUI

Spyder scientific development environment

```
$ spyder
```

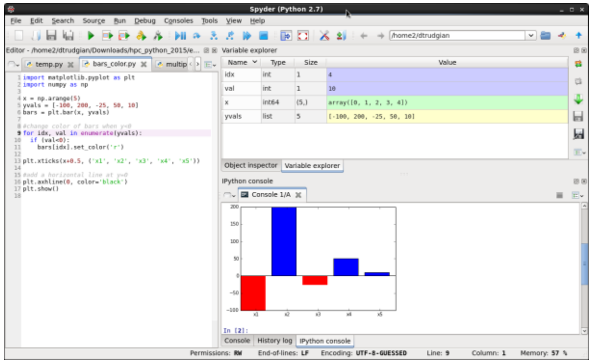
Jupyter Notebook on Demand

https://portal.biohpc.swmed.edu/terminal/ondemand_jupyter/

JupyterLab on Demand

https://portal.biohpc.swmed.edu/terminal/ondemand_jupyterlab/

** training on Jupyter Notebook is on 5/19/2021



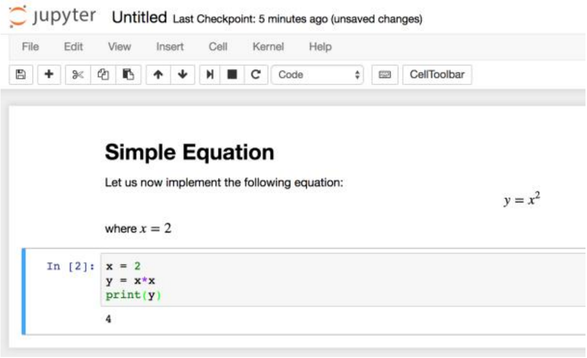
```
import matplotlib.pyplot as plt
import numpy as np

x = np.arange(5)
yvals = [-100, 200, -25, 50, 10]
bars = plt.bar(x, yvals)

#change color of bars when y<0
for idx, val in enumerate(yvals):
  if (val<0):
    bars[idx].set_color('r')

plt.xticks(x+0.5, ('x1', 'x2', 'x3', 'x4', 'x5'))

#add a horizontal line at y=0
plt.axhline(0, color='black')
plt.show()
```

Name	Type	Size	Value
idx	int	1	4
val	int	1	10
x	int64	(5,)	array([0, 1, 2, 3, 4])
yvals	list	5	[-100, 200, -25, 50, 10]

```
In [2]:
```

Permissions: RW End-of-lines: LF Encoding: UTF-8-GUESSED Line: 9 Column: 1 Memory: 57 %

Spyder



jupyter
Untitled Last Checkpoint: 5 minutes ago (unsaved changes)
File Edit View Insert Cell Kernel Help
Code
CellToolbar

Simple Equation

Let us now implement the following equation:

$$y = x^2$$

where $x = 2$

```
In [2]: x = 2
        y = x*x
        print(y)

        4
```

Jupyter Notebook

Conda – install miniconda

- Anaconda Not recommended

  Hundreds of scientific packages automatically installed at once

  Too many small files, and 4G space
- Miniconda Recommended

  Python, conda and some essential packages, 350M space

```
# Download miniconda script
wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh

# Run script
bash Miniconda3-latest-Linux-x86_64.sh
# Follow prompts, accept license (yes)
# Specify install location if needed
# initialize Miniconda3 (no)

# Activate environment
conda activate /home2/<username>/miniconda3
# check all installed packages
conda list
```

https://docs.conda.io/en/latest/miniconda.html

Conda – Create Your Own Environment

The main module installation must be stable

*We **won’t** update packages in it frequently.*

The conda tool lets you create your own environments with versions you need

Stored in $HOME/.conda $HOME 50G space

```
# Create a new environment with package biopython
conda create -n test1 biopython

# See information about environments available
conda info -e

# Start using this environment*
conda activate test1

# Back to default environment*
conda deactivate

# Create a new environment to a different directory
conda create --prefix /project/<dept>/<lab>/<user>/test3 biopython
```

Conda – Create Your Own Environment

```
# Create a minimal environment with specific python and numpy
# Won't install all of the conda package set
conda create -n test2 python=3.6.4 numpy=1.16

# Start using the environment
conda activate test2

# Add more package rpy2 to this active environment
conda install rpy2

# Update the numpy package to the latest version
conda update numpy

# Install a non-conda package using pip
conda search planemo
conda install pip
pip install planemo
```

More user guide:

Managing environments — conda 4.9.2 documentation

Issue of conda environments

- Too many small files!
  anaconda: ~165,000 files, 4G. Do NOT install anaconda
  test2 (python and numpy only): ~15,000 files on BioHPC
- 165,000 files * ~ 1,000 biohpc users = 165,000,000 files.
- Pressure on BioHPC storage system

Solutions:

a) Create shared conda environment to lab shared directories.
   - All users can read/write. Be careful! Any user can update the lab packages
   - Only specific user can change it. An admin to maintain lab conda env

b) Popular python package as BioHPC module. Request BioHPC team to install.

c) BioHPC is going to provide an option to use Singularity containers with overlay filesystem for conda.

Always consult with BioHPC first if want to install large conda environment, set up lab conda environment

Send email to biohpc-help@utsouthwestern.edu

Challenge in Python

Python is slooooooow…..

Trades execution speed for development speed.

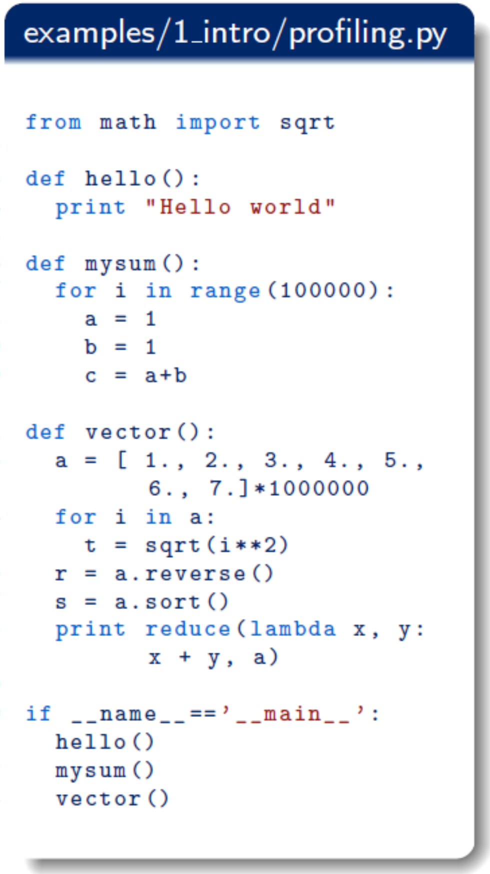
Solution: Move critical portions closer to machine code.

- Directly call C code - Cython
- Use modules built on optimized, compiled code.
  e.g. NumPy builds on BLAS / LAPACK

Profiling – method calls

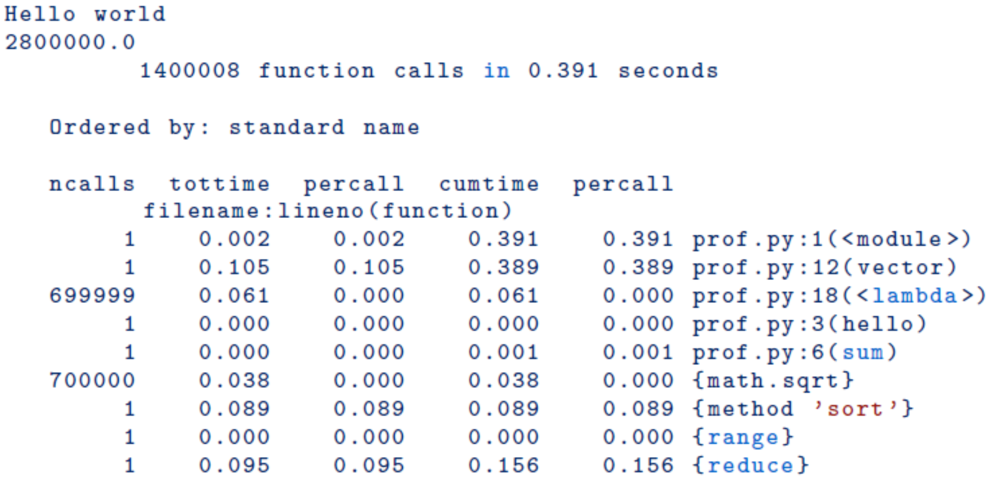
profiling.py

```
python -m cProfile [-o output_file] [-s sort_order] script.py
```



examples/1_intro/profiling.py

```
 1  from math import sqrt
 2
 3  def hello():
 4    print "Hello world"
 5
 6  def mysum():
 7    for i in range(100000):
 8      a = 1
 9      b = 1
10      c = a+b
11
12  def vector():
13    a = [ 1., 2., 3., 4., 5.,
          6., 7.]*1000000
14    for i in a:
15      t = sqrt(i**2)
16    r = a.reverse()
17    s = a.sort()
18    print reduce(lambda x, y:
          x + y, a)
19
20  if __name__=='__main__':
21    hello()
22    mysum()
23    vector()
```



```
Hello world
2800000.0
         1400008 function calls in 0.391 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall
        filename:lineno(function)
        1    0.002    0.002    0.391    0.391 prof.py:1(<module>)
        1    0.105    0.105    0.389    0.389 prof.py:12(vector)
   699999    0.061    0.000    0.061    0.000 prof.py:18(<lambda>)
        1    0.000    0.000    0.000    0.000 prof.py:3(hello)
        1    0.000    0.000    0.001    0.001 prof.py:6(sum)
   700000    0.038    0.000    0.038    0.000 {math.sqrt}
        1    0.089    0.089    0.089    0.089 {method 'sort'}
        1    0.000    0.000    0.000    0.000 {range}
        1    0.095    0.095    0.156    0.156 {reduce}
```

Profiling – line by line

test_prof1.py

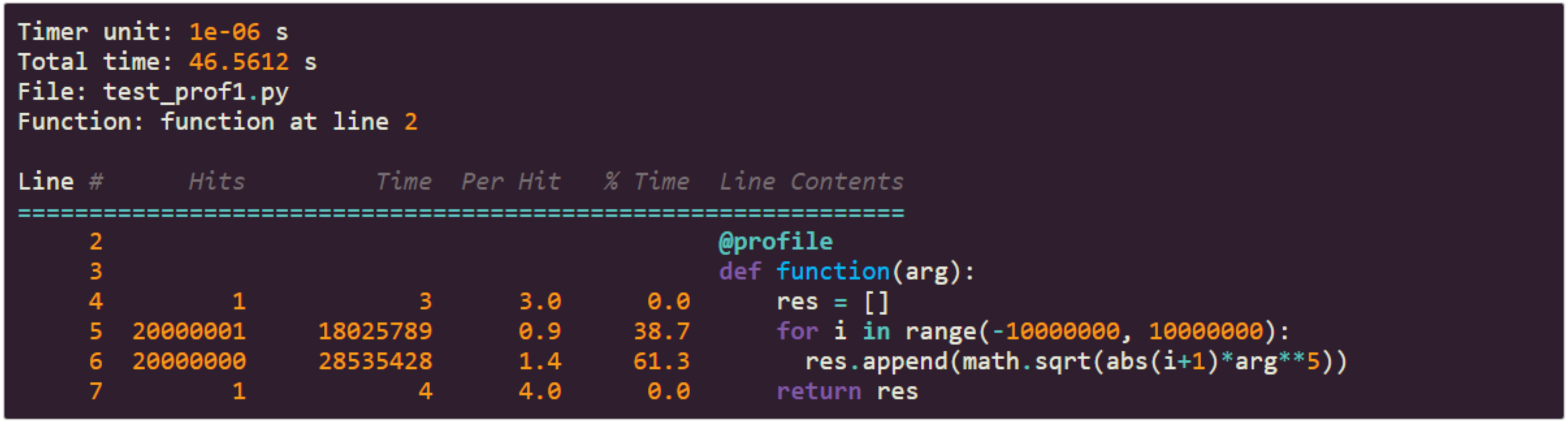
In a custom environment install module:

```
$ conda install line_profiler
```

Add `@profile` decorator to functions in code that you want to profile

Run the profiler:

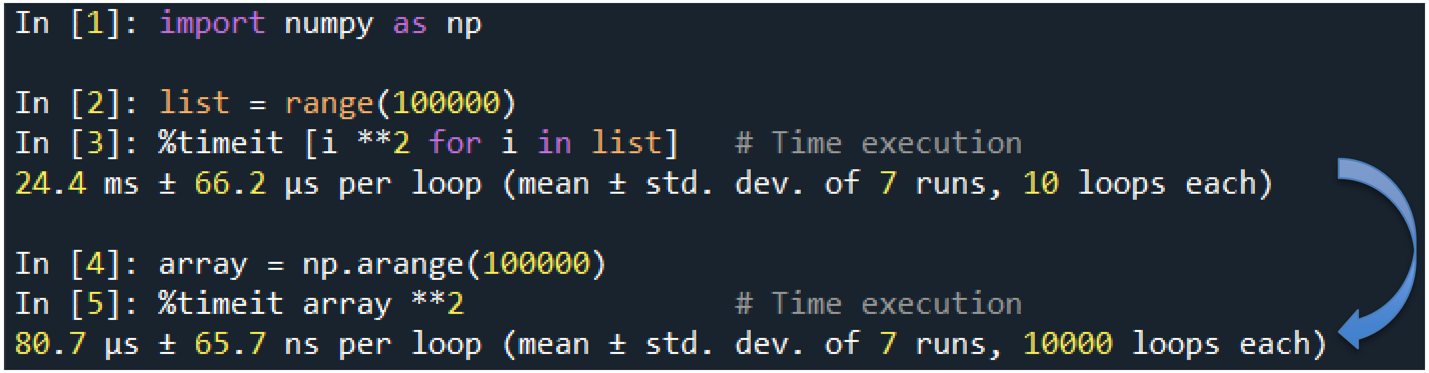
```
$ kernprof -l -v test_prof1.py
```



```
Timer unit: 1e-06 s
Total time: 46.5612 s
File: test_prof1.py
Function: function at line 2

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     2                                           @profile
     3                                           def function(arg):
     4         1            3      3.0      0.0      res = []
     5  20000001     18025789      0.9     38.7      for i in range(-10000000, 10000000):
     6  20000000     28535428      1.4     61.3        res.append(math.sqrt(abs(i+1)*arg**5))
     7         1            4      4.0      0.0      return res
```

NumPy

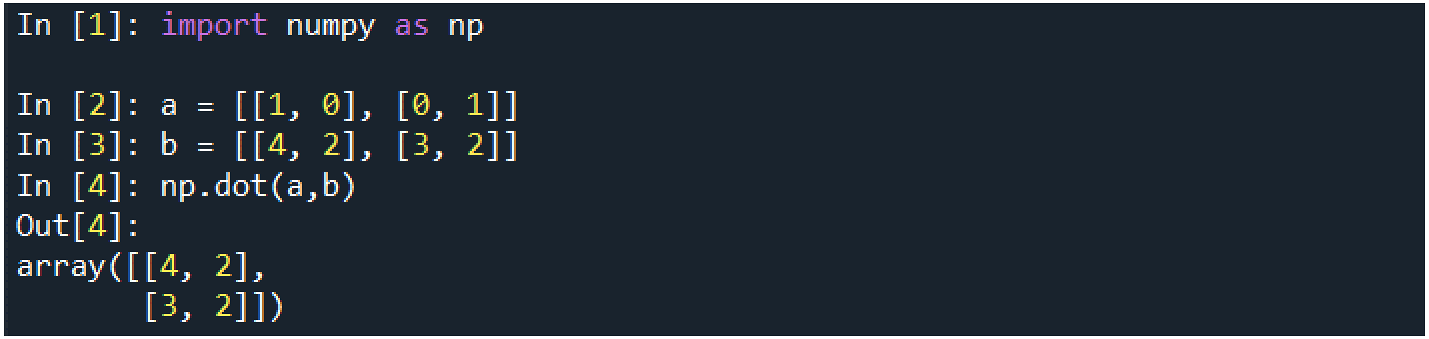
NumPy performs (multi-dimensional) array arithmetic *much* faster than native python objects, by using low-level contiguous arrays and compiled libraries:



```
In [1]: import numpy as np

In [2]: list = range(100000)
In [3]: %timeit [i **2 for i in list]    # Time execution
24.4 ms ± 66.2 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)

In [4]: array = np.arange(100000)
In [5]: %timeit array **2                # Time execution
80.7 µs ± 65.7 ns per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

300 x faster

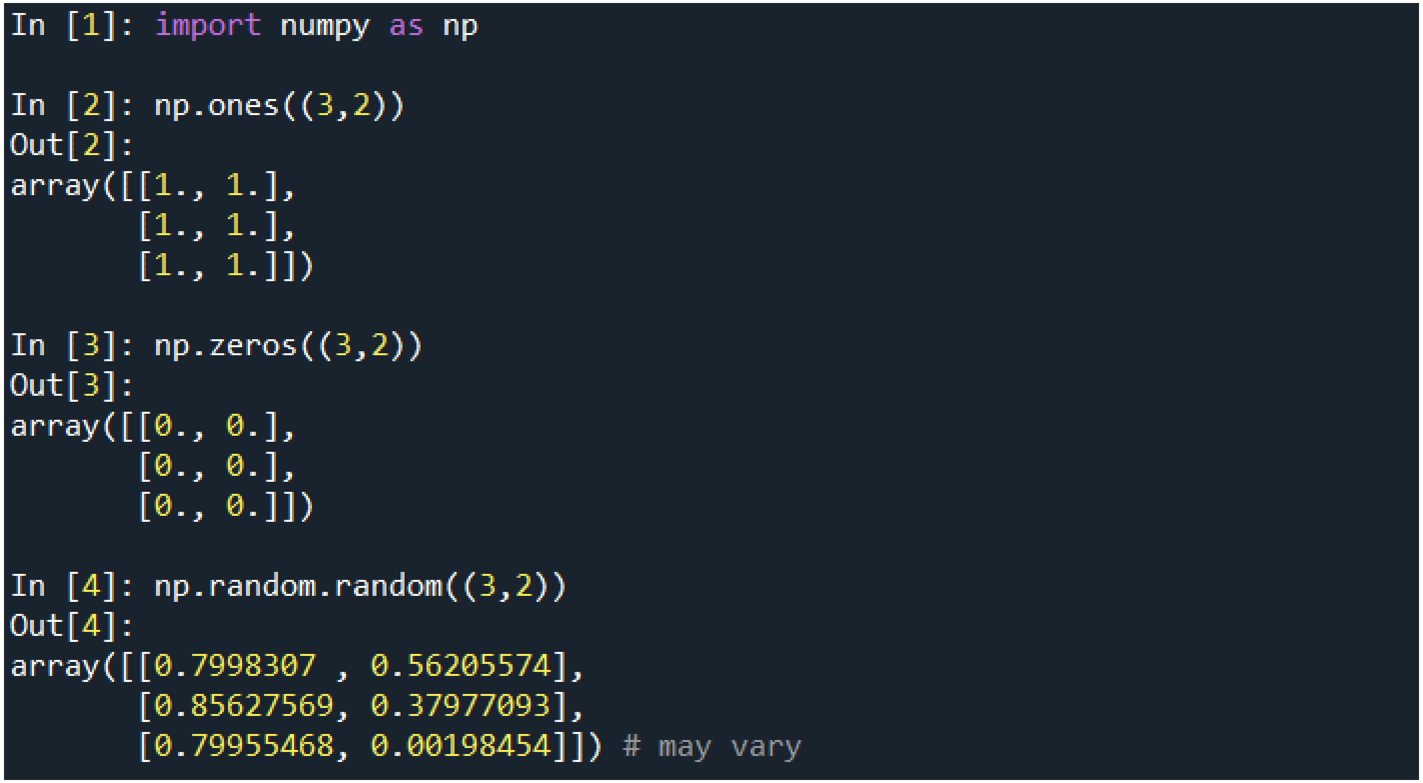
The Linear Algebra module of NumPy offers various methods to apply linear algebra on any Numpy array.



```
In [1]: import numpy as np

In [2]: a = [[1, 0], [0, 1]]
In [3]: b = [[4, 2], [3, 2]]
In [4]: np.dot(a,b)
Out[4]:
array([[4, 2],
       [3, 2]])
```

NumPy

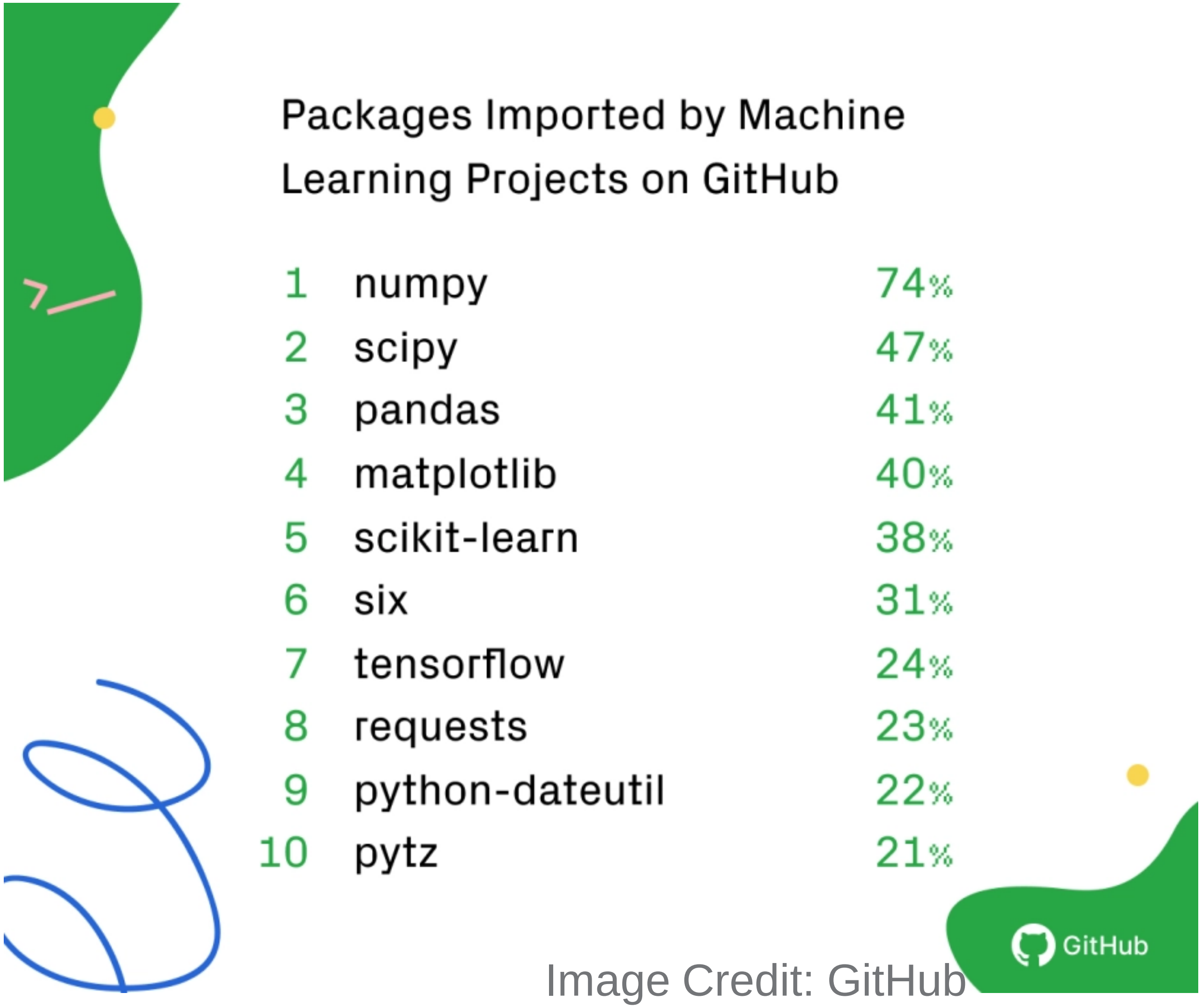
Create array and manipulation



```
In [1]: import numpy as np

In [2]: np.ones((3,2))
Out[2]:
array([[1., 1.],
       [1., 1.],
       [1., 1.]])

In [3]: np.zeros((3,2))
Out[3]:
array([[0., 0.],
       [0., 0.],
       [0., 0.]])

In [4]: np.random.random((3,2))
Out[4]:
array([[0.7998307 , 0.56205574],
       [0.85627569, 0.37977093],
       [0.79955468, 0.00198454]]) # may vary
```

https://numpy.org/learn/

More popular packages



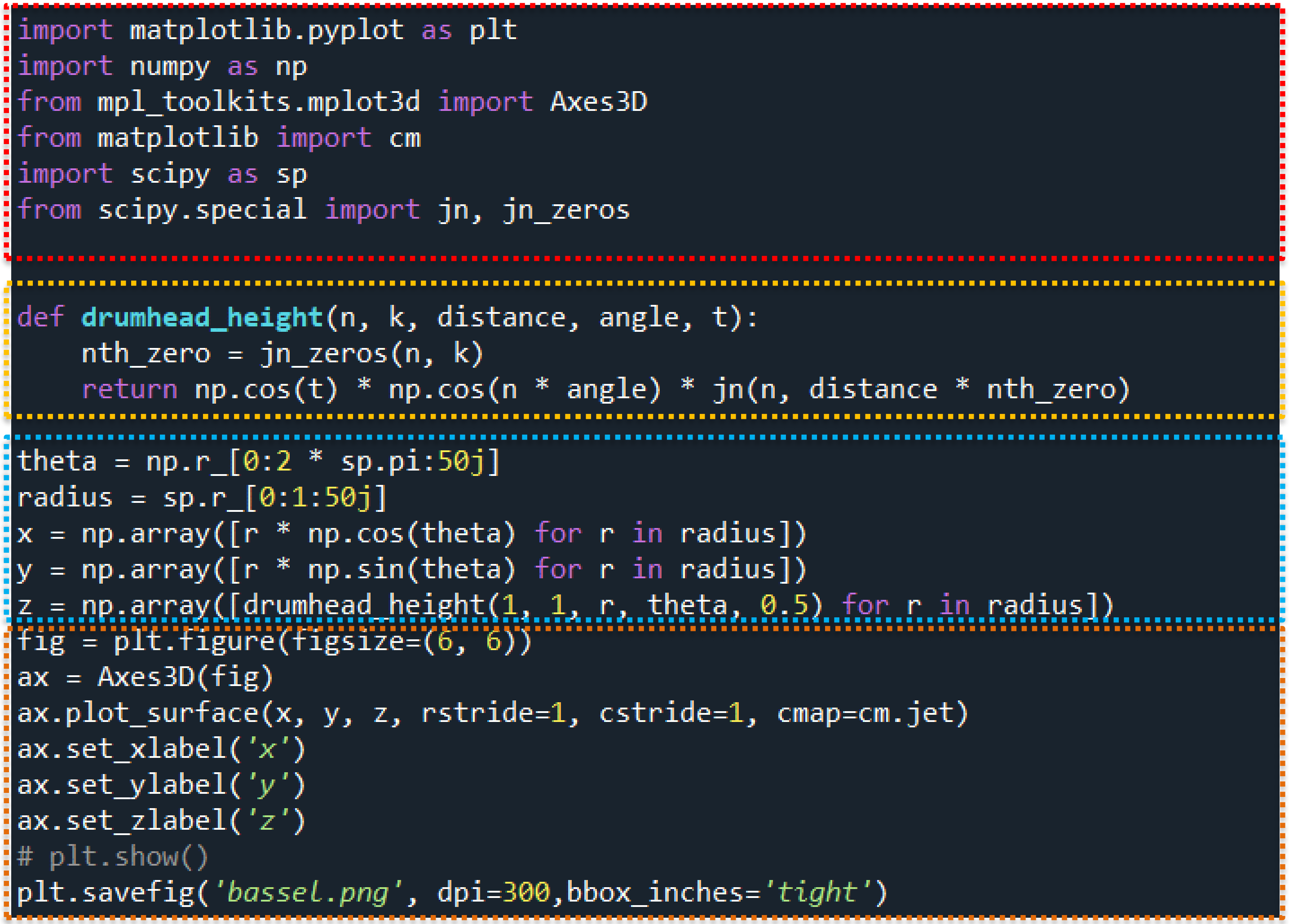
Packages Imported by Machine Learning Projects on GitHub

1	numpy	74%
2	scipy	47%
3	pandas	41%
4	matplotlib	40%
5	scikit-learn	38%
6	six	31%
7	tensorflow	24%
8	requests	23%
9	python-dateutil	22%
10	pytz	21%

Image Credit: GitHub

SciPy & Matplotlib

Bessel_2021.py



```
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
import scipy as sp
from scipy.special import jn, jn_zeros
```

# comes your imports

```
def drumhead_height(n, k, distance, angle, t):
    nth_zero = jn_zeros(n, k)
    return np.cos(t) * np.cos(n * angle) * jn(n, distance * nth_zero)
```

# User defined function(s)

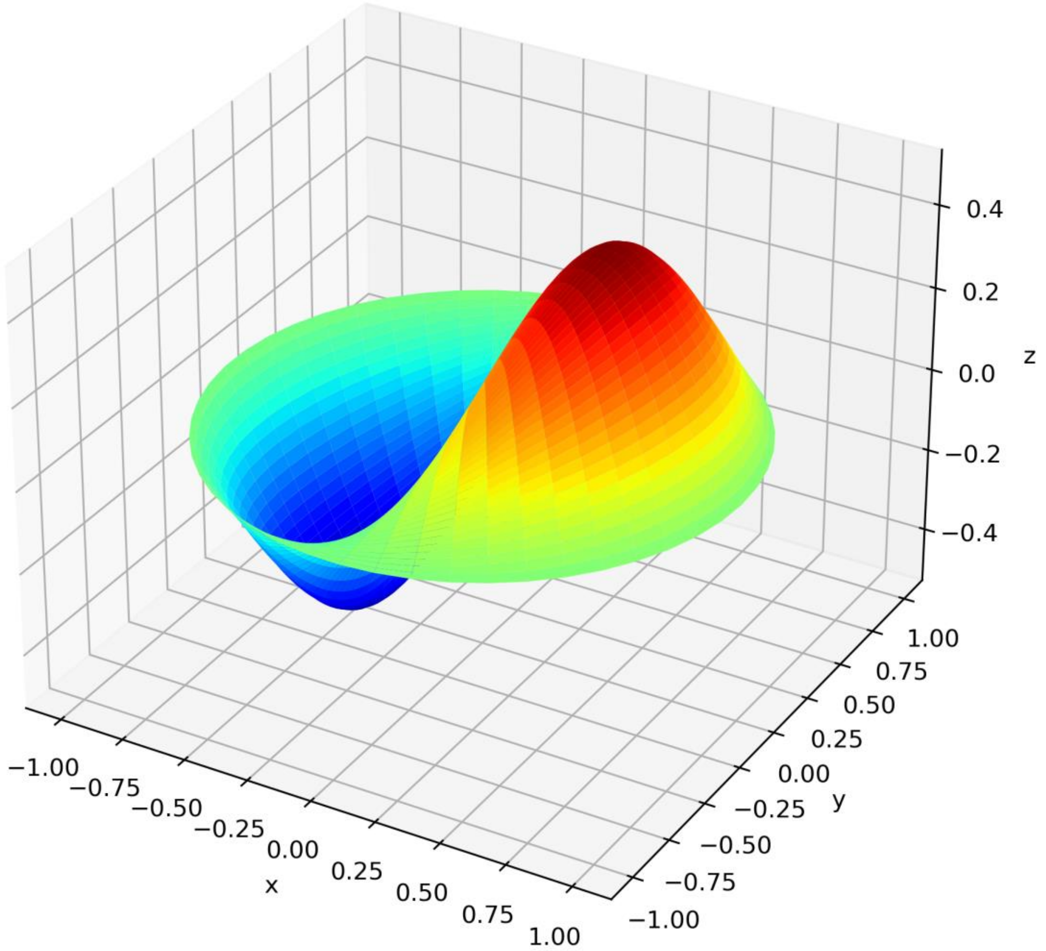
```
theta = np.r_[0:2 * sp.pi:50j]
radius = sp.r_[0:1:50j]
x = np.array([r * np.cos(theta) for r in radius])
y = np.array([r * np.sin(theta) for r in radius])
z = np.array([drumhead_height(1, 1, r, theta, 0.5) for r in radius])
```

# calculation

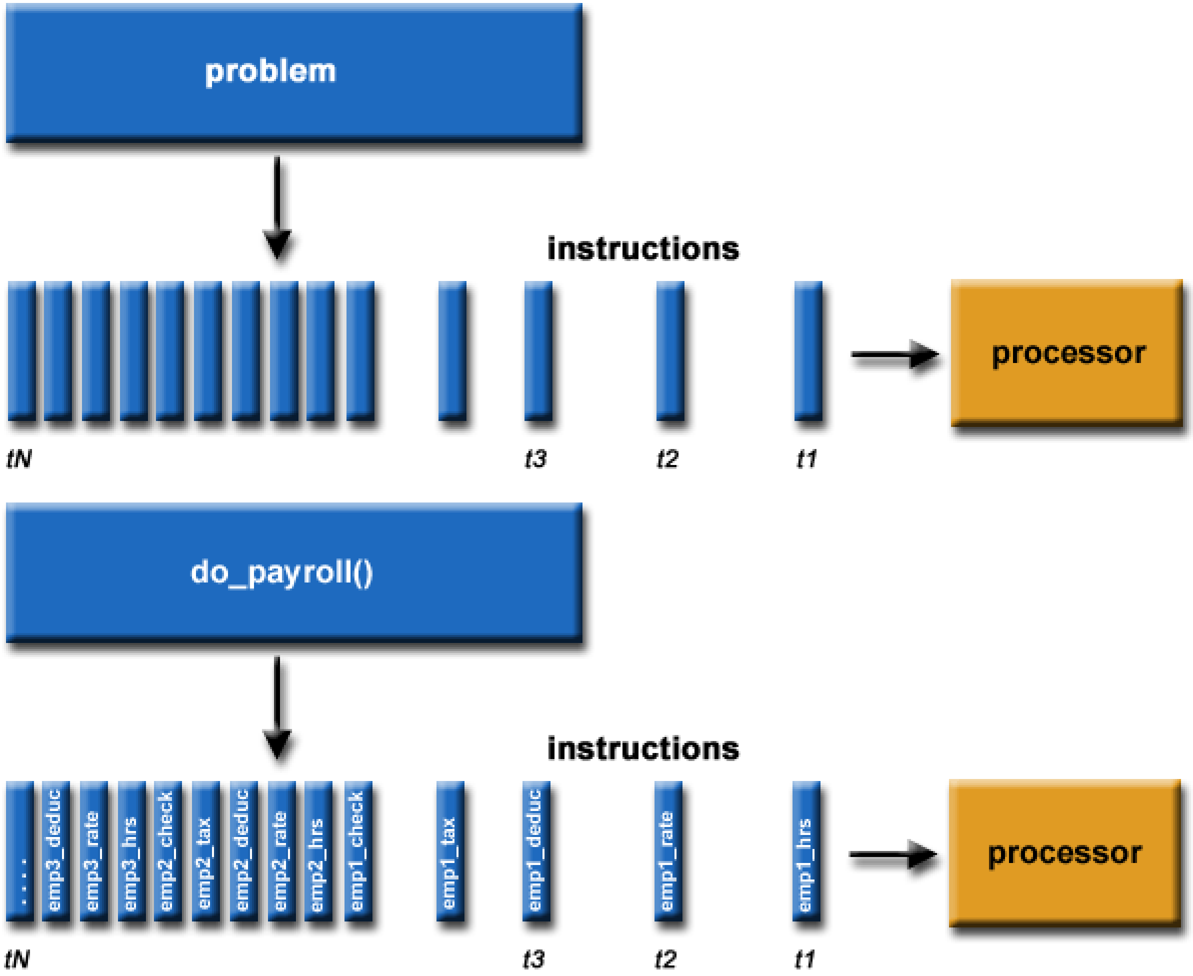
```
fig = plt.figure(figsize=(6, 6))
ax = Axes3D(fig)
ax.plot_surface(x, y, z, rstride=1, cstride=1, cmap=cm.jet)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
# plt.show()
plt.savefig('bassel.png', dpi=300,bbox_inches='tight')
```

# visualization

SciPy & Matplotlib



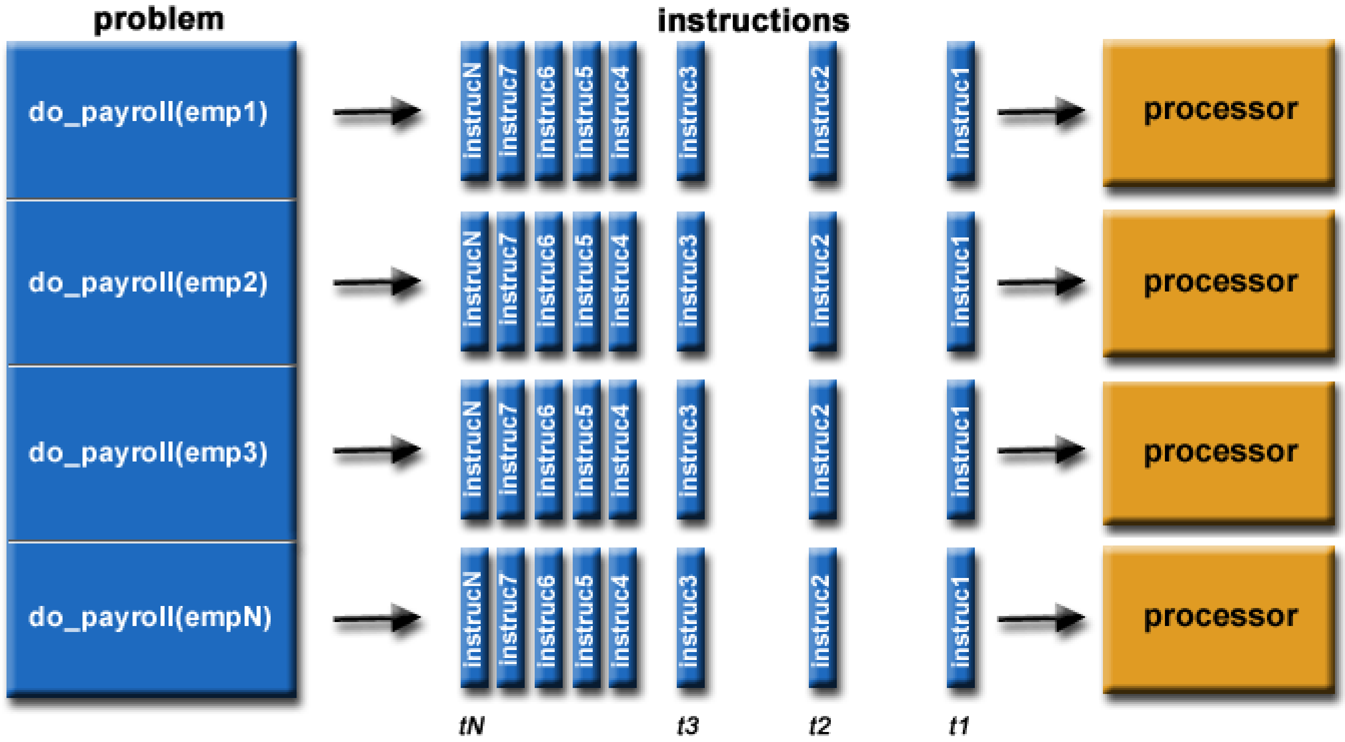
Serial Computing and Parallel Computing



Serial Computing

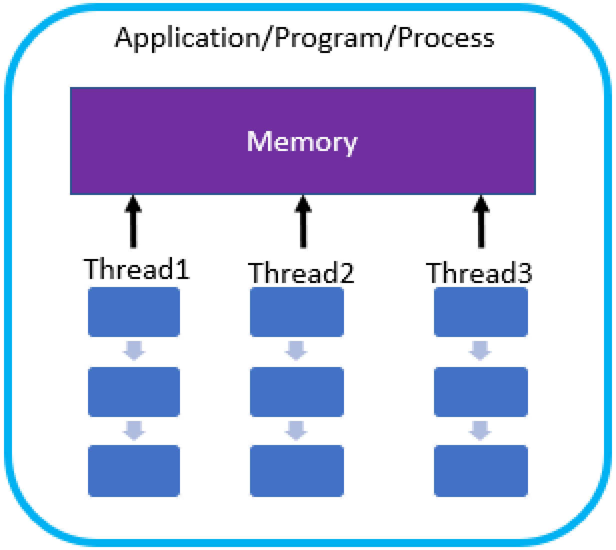
[Introduction to Parallel Computing Tutorial | High Performance Computing (llnl.gov)]

Serial Computing and Parallel Computing



Parallel Computing:
Breaking a problem into multiple pieces and processing each piece in parallel through multiple processors

Multi-Threading vs. Multi-Processing



Application/Program/Process

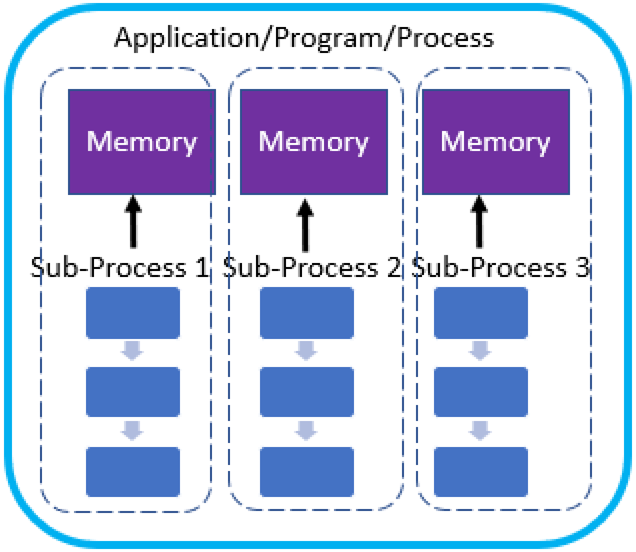
Memory

Thread1

Thread2

Thread3

Multi-Threading



Application/Program/Process

Memory

Memory

Memory

Sub-Process 1

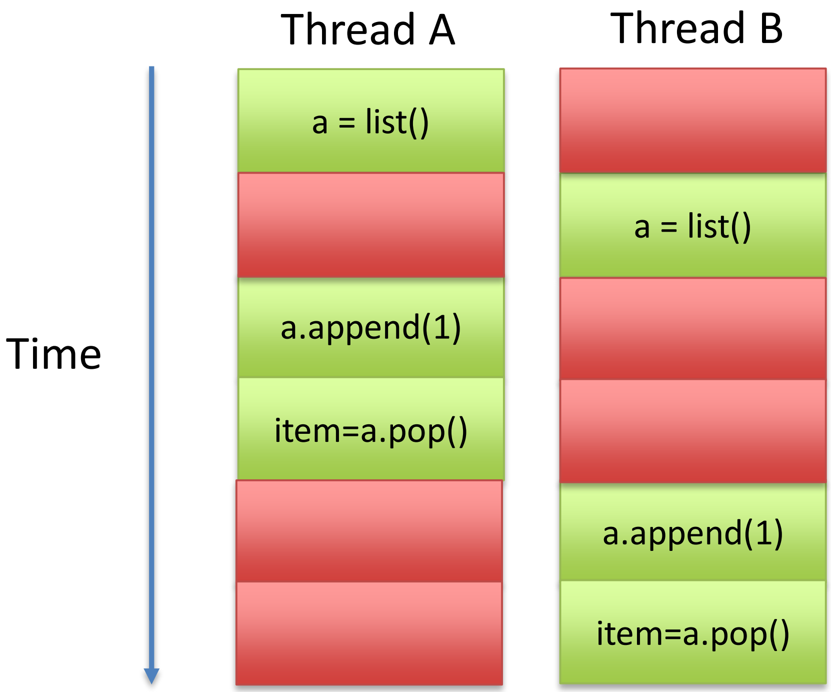
Sub-Process 2

Sub-Process 3

Multi-processing

Challenge in Python

Global Interpreter Lock



Time

Thread A	Thread B
a = list()	🟥
🟥	a = list()
a.append(1)	🟥
item=a.pop()	🟥
🟥	a.append(1)
🟥	item=a.pop()

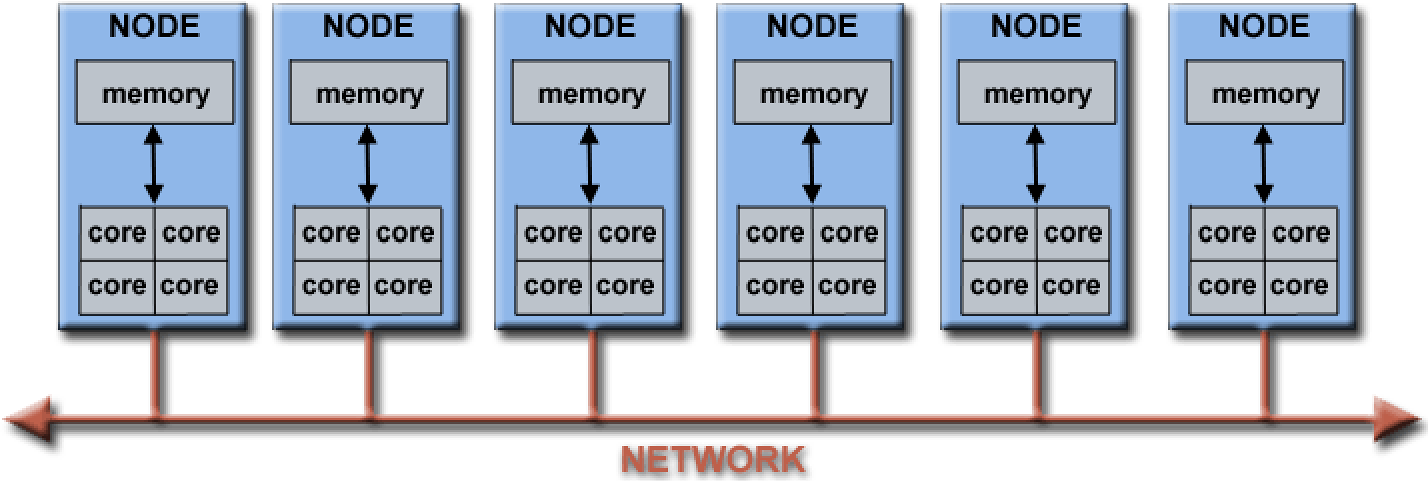
Can create many threads, but only runs 1 thread at a time.

Solution – multiple processes

Parallel Python Computing on BioHPC

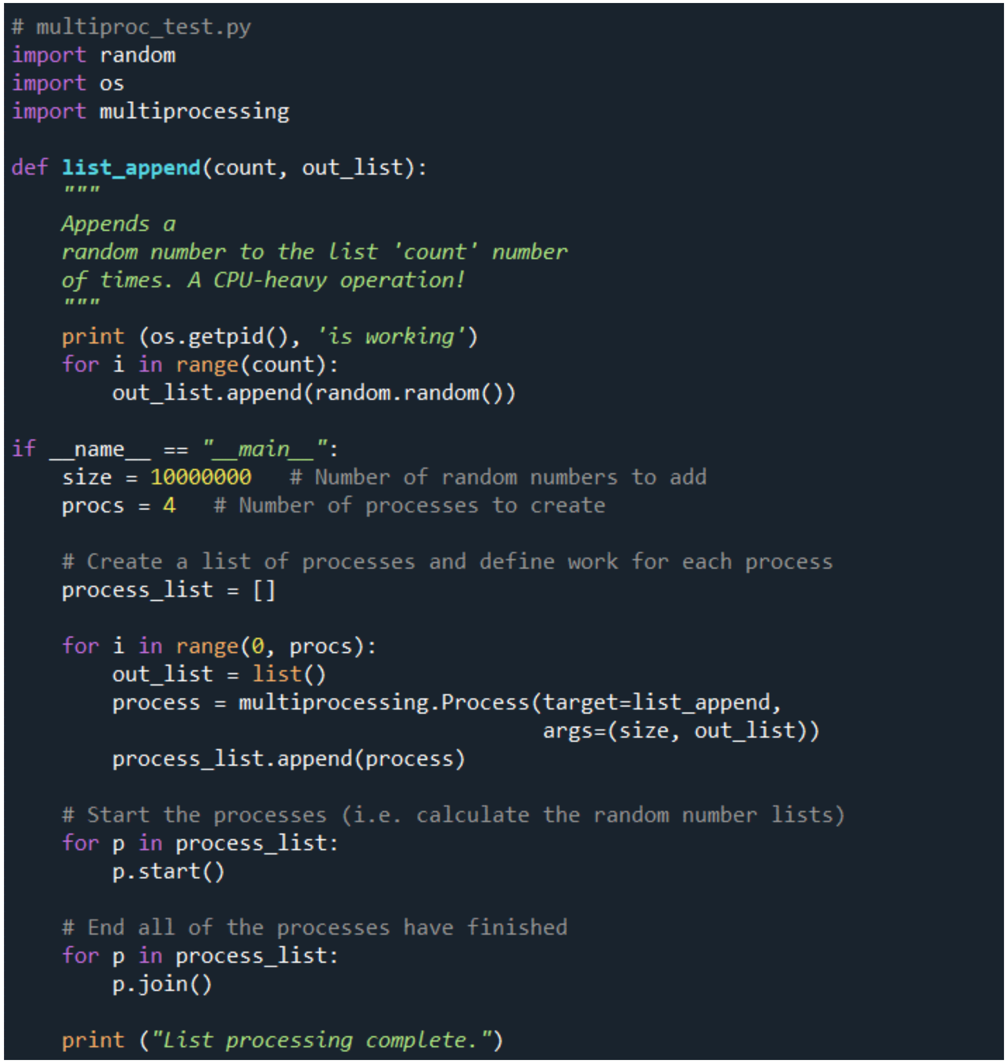
- BioHPCs employ often 2-4 server-grade CPUs per node
  - 8 – 16 processor cores per CPU
  - Shared memory on each node for all processors

- Distributed memory architecture
  - Nodes are connected via a high-speed network
  - Memory is shared between nodes through some API
    - MPI is most commonly used



Multiprocessing – Direct Creation & Management

multiproc_test_2021.py



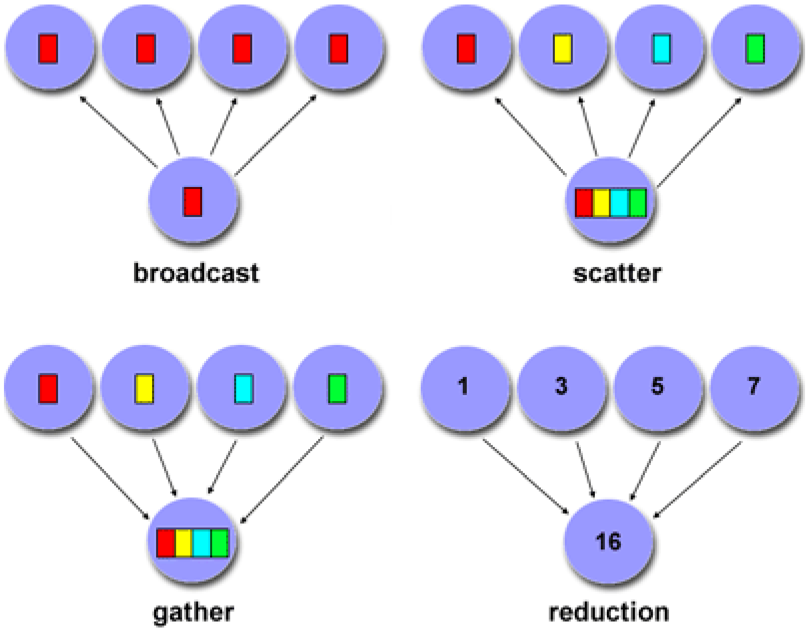
```python
# multiproc_test.py
import random
import os
import multiprocessing

def list_append(count, out_list):
    """
    Appends a
    random number to the list 'count' number
    of times. A CPU-heavy operation!
    """
    print (os.getpid(), 'is working')
    for i in range(count):
        out_list.append(random.random())

if __name__ == "__main__":
    size = 10000000   # Number of random numbers to add
    procs = 4   # Number of processes to create

    # Create a list of processes and define work for each process
    process_list = []

    for i in range(0, procs):
        out_list = list()
        process = multiprocessing.Process(target=list_append,
                                          args=(size, out_list))
        process_list.append(process)

    # Start the processes (i.e. calculate the random number lists)
    for p in process_list:
        p.start()

    # End all of the processes have finished
    for p in process_list:
        p.join()

    print ("List processing complete.")
```

```
Output[]:
57526 is working
57527 is working
57532 is working
57545 is working
List processing complete.
```

MPI

A interface for parallel computation using message passing between processes

Small set of instructions, but quite complicate to use

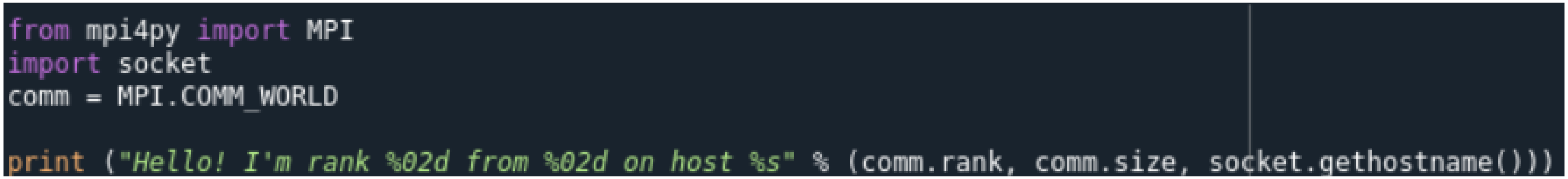


Mpi4py – MPI wrappers for python

Hello_mpi_2021.py

Install the module

```
(test2)[ ] $ conda install mpi4py
```



```python
from mpi4py import MPI
import socket
comm = MPI.COMM_WORLD

print ("Hello! I'm rank %02d from %02d on host %s" % (comm.rank, comm.size, socket.gethostname()))
```

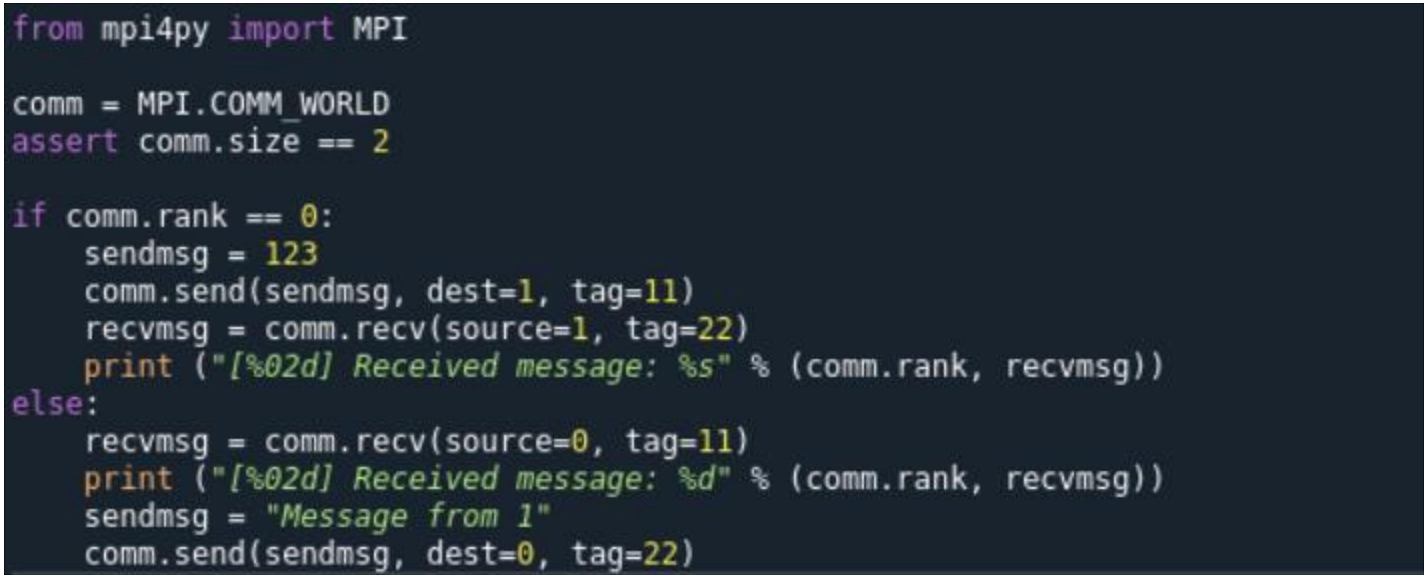
Run the code

```
(test2)[ ] $ mpirun -n 4 python hello_mpi_2021.py

Hello! I'm rank 03 from 04 on host NucleusA140
Hello! I'm rank 00 from 04 on host NucleusA140
Hello! I'm rank 01 from 04 on host NucleusA140
Hello! I'm rank 02 from 04 on host NucleusA140
```

https://mpi4py.readthedocs.io/en/stable/tutorial.html

mpi4py – Communication of python objects

P2p_2021.py



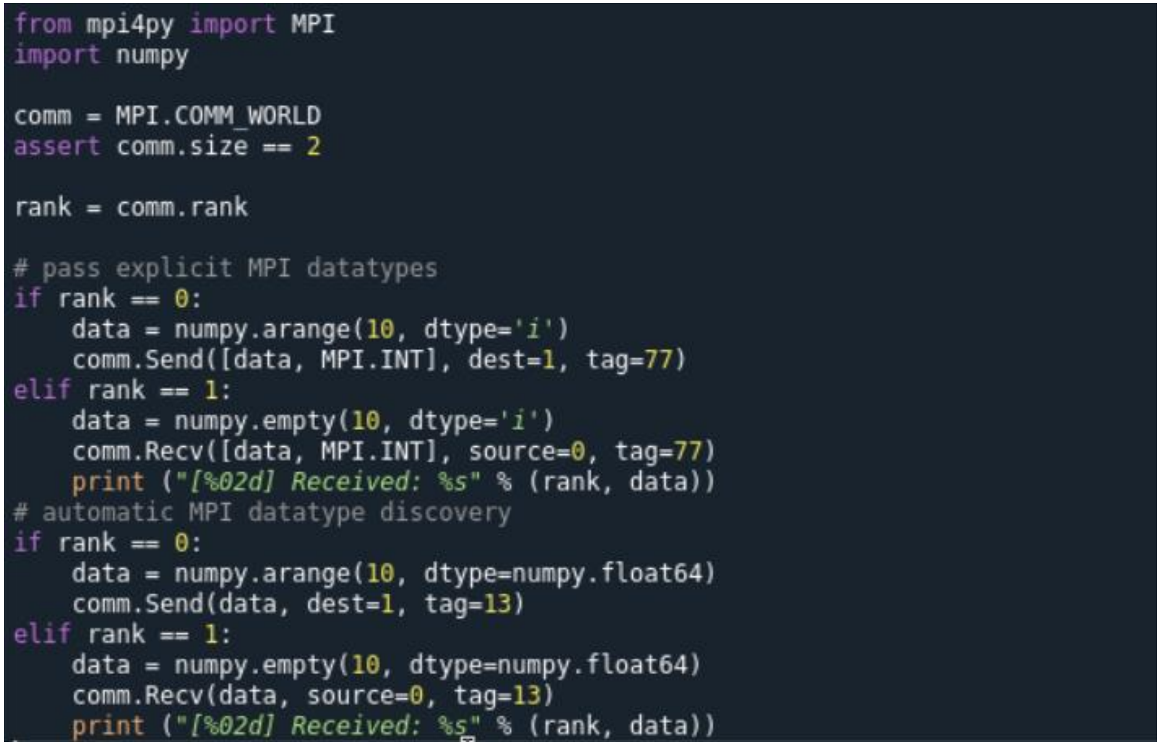
```python
from mpi4py import MPI

comm = MPI.COMM_WORLD
assert comm.size == 2

if comm.rank == 0:
    sendmsg = 123
    comm.send(sendmsg, dest=1, tag=11)
    recvmsg = comm.recv(source=1, tag=22)
    print ("[%02d] Received message: %s" % (comm.rank, recvmsg))
else:
    recvmsg = comm.recv(source=0, tag=11)
    print ("[%02d] Received message: %d" % (comm.rank, recvmsg))
    sendmsg = "Message from 1"
    comm.send(sendmsg, dest=0, tag=22)
```

```
(test2) [ ] $ mpirun -n 2 python p2p.py

[01] Received message: 123
[00] Received message: Message from 1
```

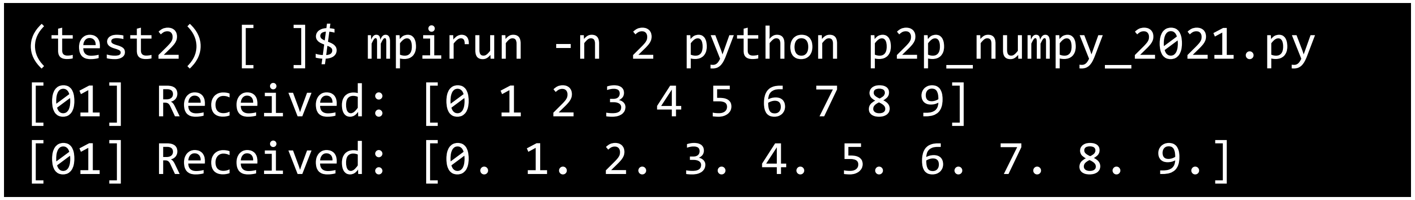
SLOW! – Python objects must be serialized & deserialized.

mpi4py – Communication of numpy arrays

p2p_numpy_2021.py



```
from mpi4py import MPI
import numpy

comm = MPI.COMM_WORLD
assert comm.size == 2

rank = comm.rank

# pass explicit MPI datatypes
if rank == 0:
    data = numpy.arange(10, dtype='i')
    comm.Send([data, MPI.INT], dest=1, tag=77)
elif rank == 1:
    data = numpy.empty(10, dtype='i')
    comm.Recv([data, MPI.INT], source=0, tag=77)
    print ("[%02d] Received: %s" % (rank, data))
# automatic MPI datatype discovery
if rank == 0:
    data = numpy.arange(10, dtype=numpy.float64)
    comm.Send(data, dest=1, tag=13)
elif rank == 1:
    data = numpy.empty(10, dtype=numpy.float64)
    comm.Recv(data, source=0, tag=13)
    print ("[%02d] Received: %s" % (rank, data))
```

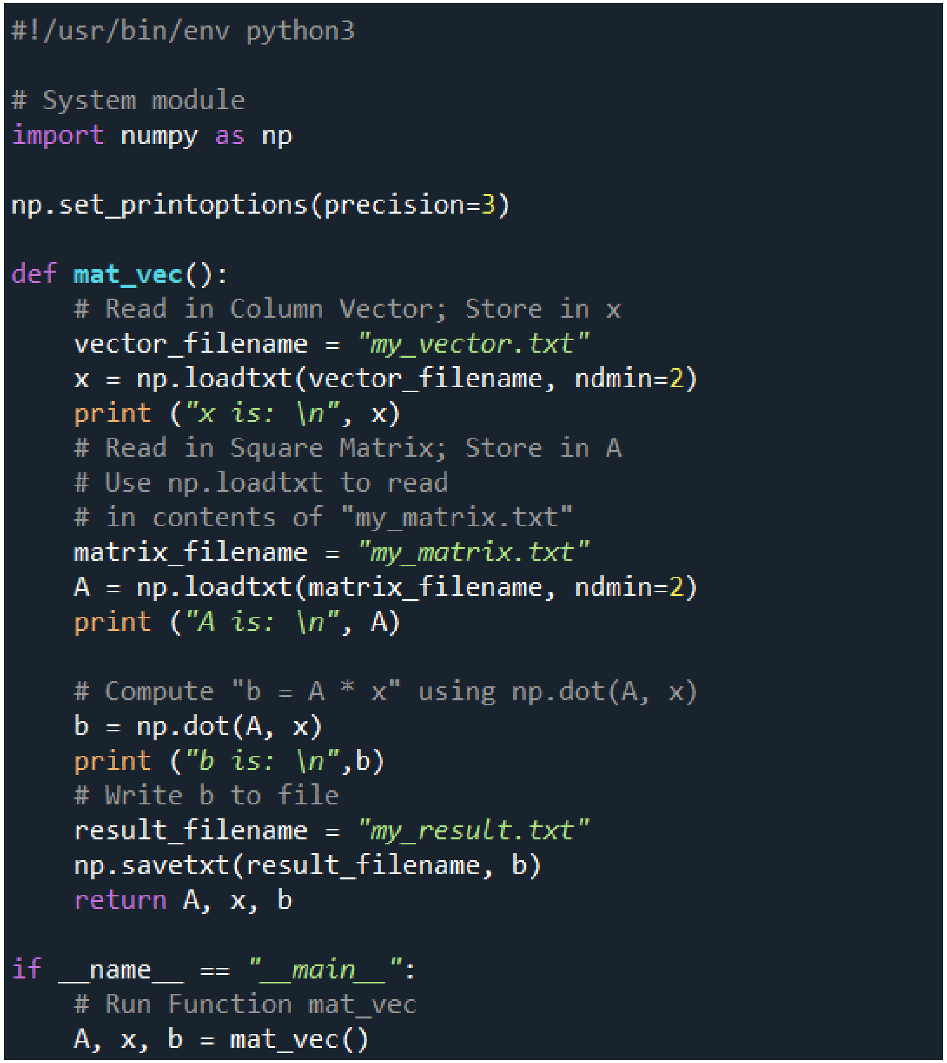


```
(test2) [ ]$ mpirun -n 2 python p2p_numpy_2021.py
[01] Received: [0 1 2 3 4 5 6 7 8 9]
[01] Received: [0. 1. 2. 3. 4. 5. 6. 7. 8. 9.]
```

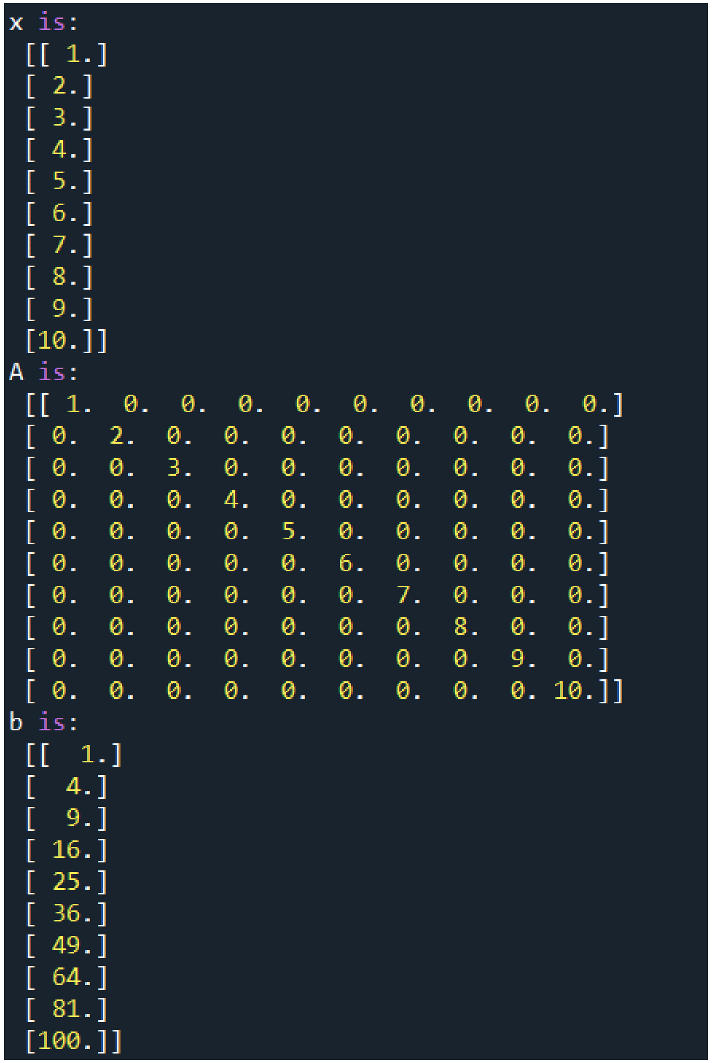
Faster – numpy arrays can be sent / received directly by the MPI layer

NumPy – Linear Algebra

matrix_vector_2021.py



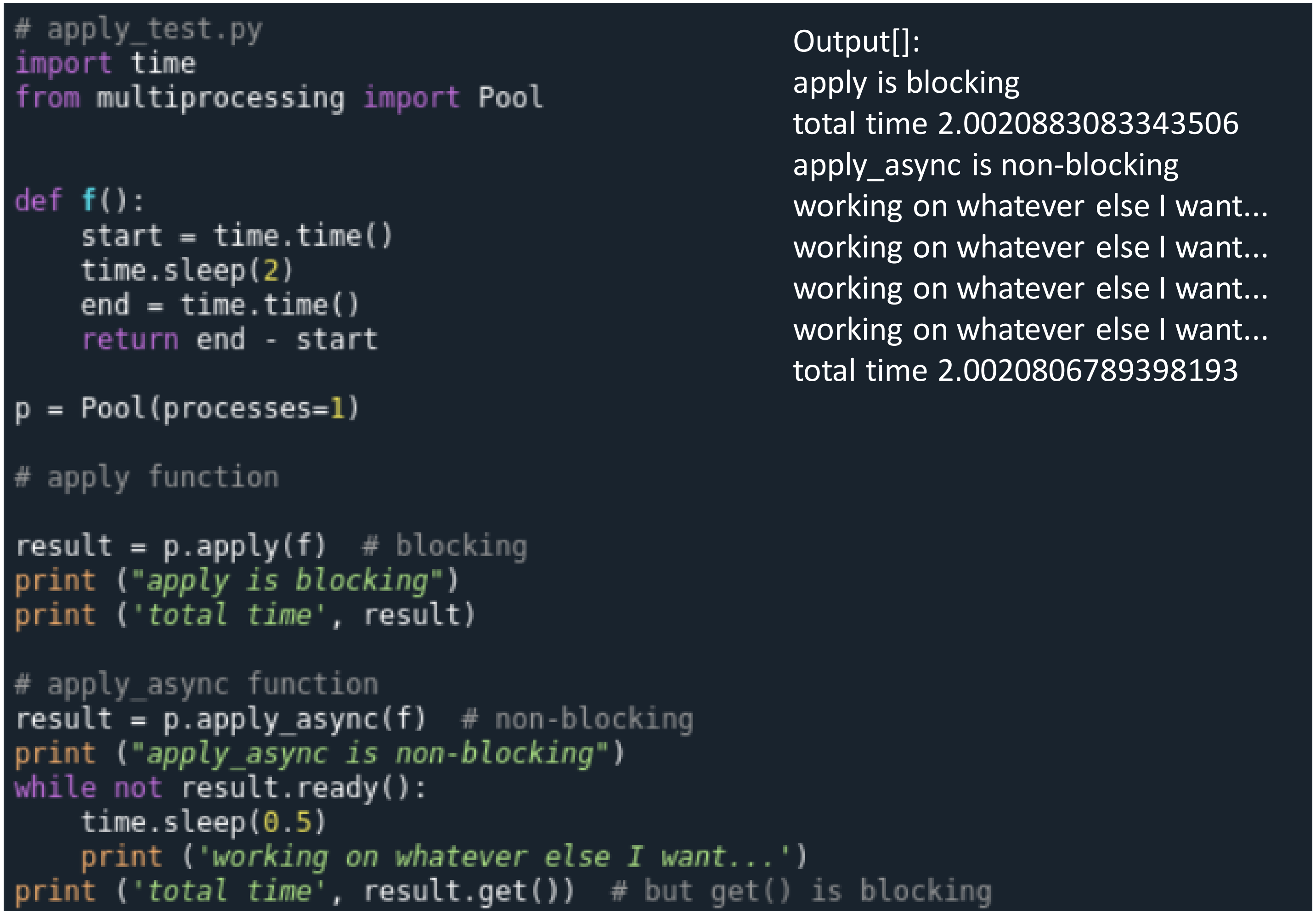
```python
#!/usr/bin/env python3

# System module
import numpy as np

np.set_printoptions(precision=3)

def mat_vec():
    # Read in Column Vector; Store in x
    vector_filename = "my_vector.txt"
    x = np.loadtxt(vector_filename, ndmin=2)
    print ("x is: \n", x)
    # Read in Square Matrix; Store in A
    # Use np.loadtxt to read
    # in contents of "my_matrix.txt"
    matrix_filename = "my_matrix.txt"
    A = np.loadtxt(matrix_filename, ndmin=2)
    print ("A is: \n", A)

    # Compute "b = A * x" using np.dot(A, x)
    b = np.dot(A, x)
    print ("b is: \n",b)
    # Write b to file
    result_filename = "my_result.txt"
    np.savetxt(result_filename, b)
    return A, x, b

if __name__ == "__main__":
    # Run Function mat_vec
    A, x, b = mat_vec()
```



```
x is:
 [[ 1.]
 [ 2.]
 [ 3.]
 [ 4.]
 [ 5.]
 [ 6.]
 [ 7.]
 [ 8.]
 [ 9.]
 [10.]]
A is:
 [[ 1.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  2.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  3.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  4.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  5.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  6.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  7.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  8.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  9.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0. 10.]]
b is:
 [[  1.]
 [  4.]
 [  9.]
 [ 16.]
 [ 25.]
 [ 36.]
 [ 49.]
 [ 64.]
 [ 81.]
 [100.]]
```

Multiprocessing – Using a process pool

apply_test_2021.py

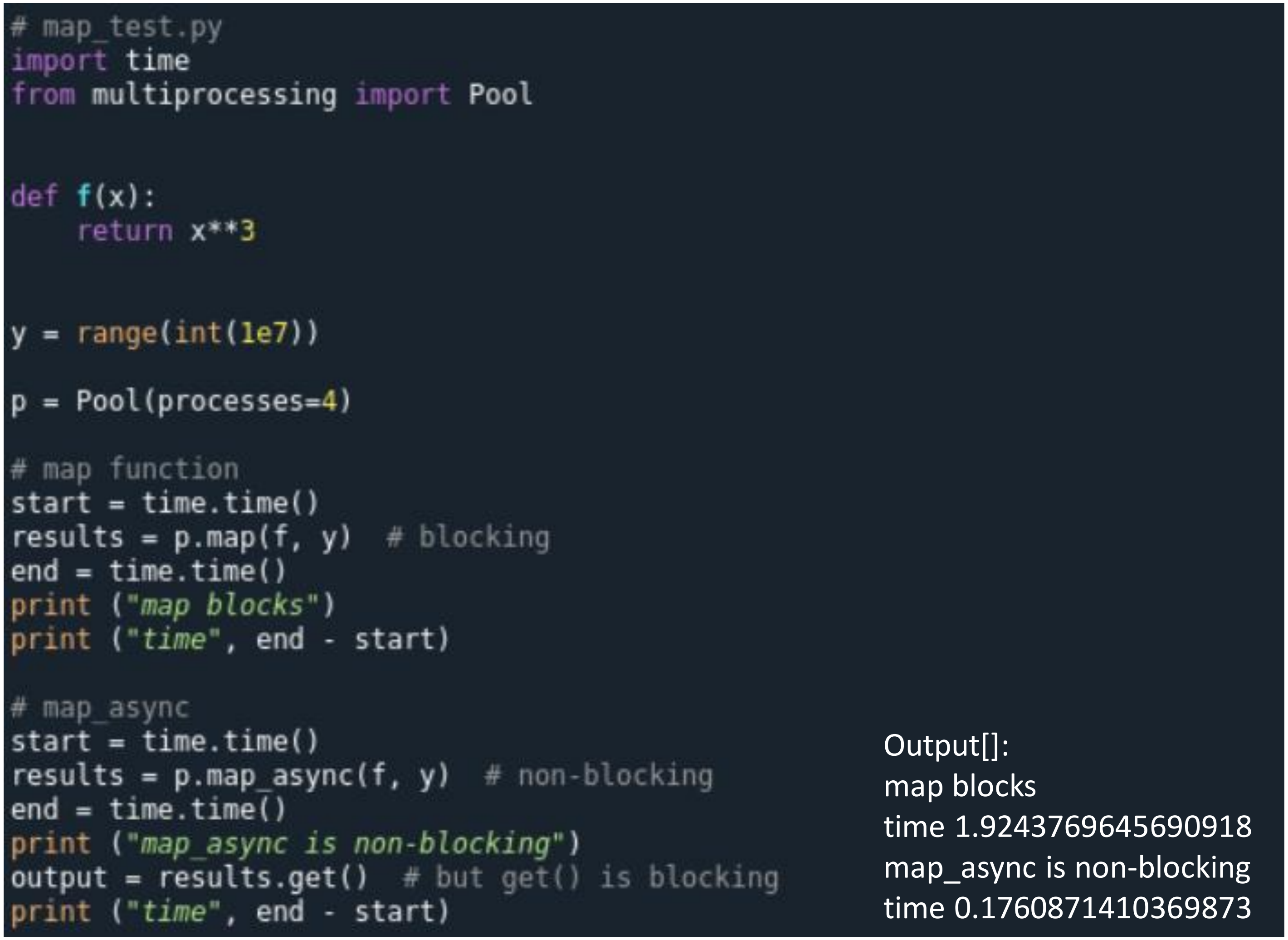


```python
# apply_test.py
import time
from multiprocessing import Pool


def f():
    start = time.time()
    time.sleep(2)
    end = time.time()
    return end - start

p = Pool(processes=1)

# apply function

result = p.apply(f)  # blocking
print ("apply is blocking")
print ('total time', result)

# apply_async function
result = p.apply_async(f)  # non-blocking
print ("apply_async is non-blocking")
while not result.ready():
    time.sleep(0.5)
    print ('working on whatever else I want...')
print ('total time', result.get())  # but get() is blocking
```

```
Output[]:
apply is blocking
total time 2.0020883083343506
apply_async is non-blocking
working on whatever else I want...
working on whatever else I want...
working on whatever else I want...
working on whatever else I want...
total time 2.0020806789398193
```

Multiprocessing – Map on an iterable object

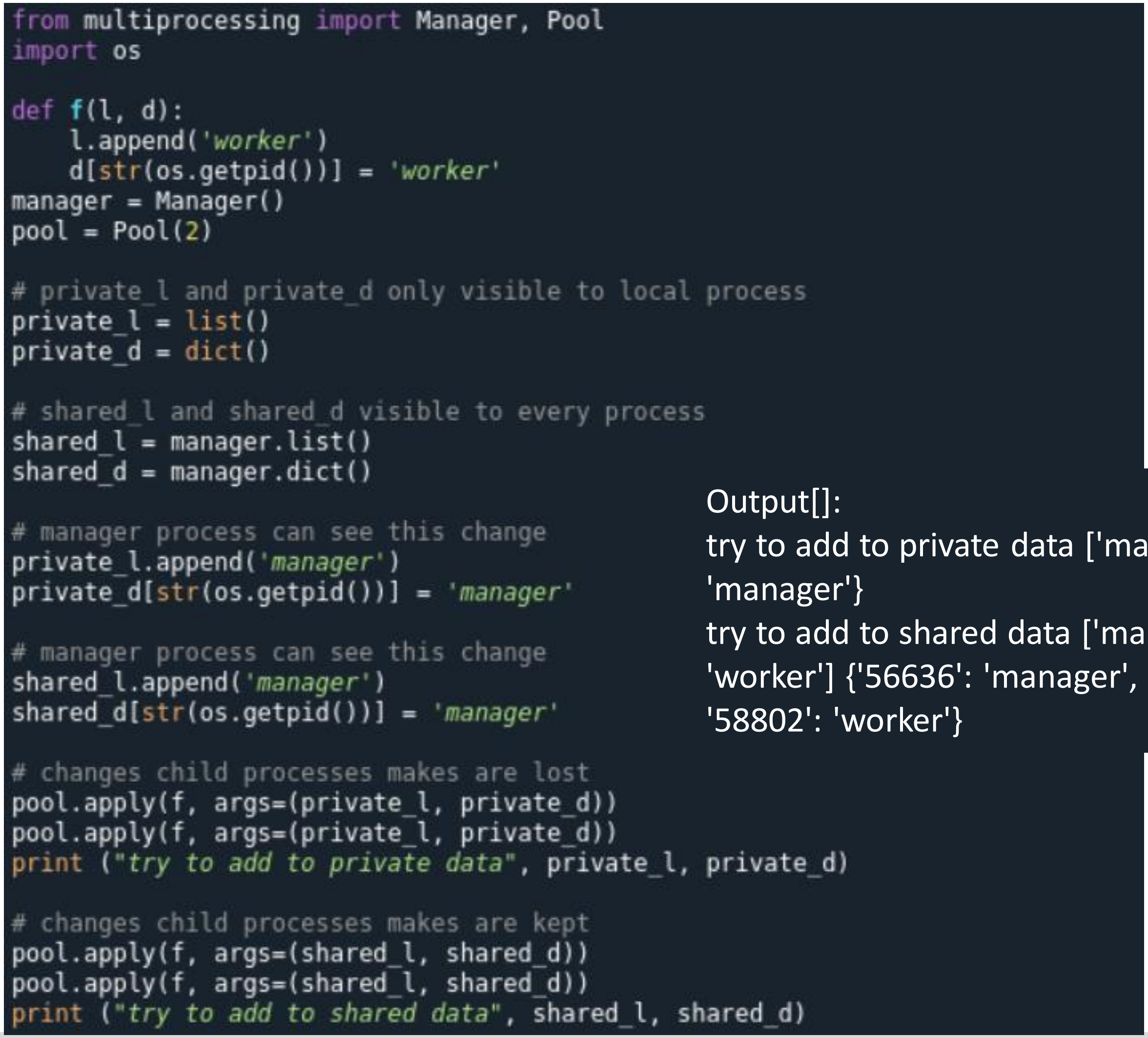
map_test_2021.py



```python
# map_test.py
import time
from multiprocessing import Pool


def f(x):
    return x**3


y = range(int(1e7))

p = Pool(processes=4)

# map function
start = time.time()
results = p.map(f, y)  # blocking
end = time.time()
print ("map blocks")
print ("time", end - start)

# map_async
start = time.time()
results = p.map_async(f, y)  # non-blocking
end = time.time()
print ("map_async is non-blocking")
output = results.get()  # but get() is blocking
print ("time", end - start)
```

```
Output[]:
map blocks
time 1.9243769645690918
map_async is non-blocking
time 0.1760871410369873
```

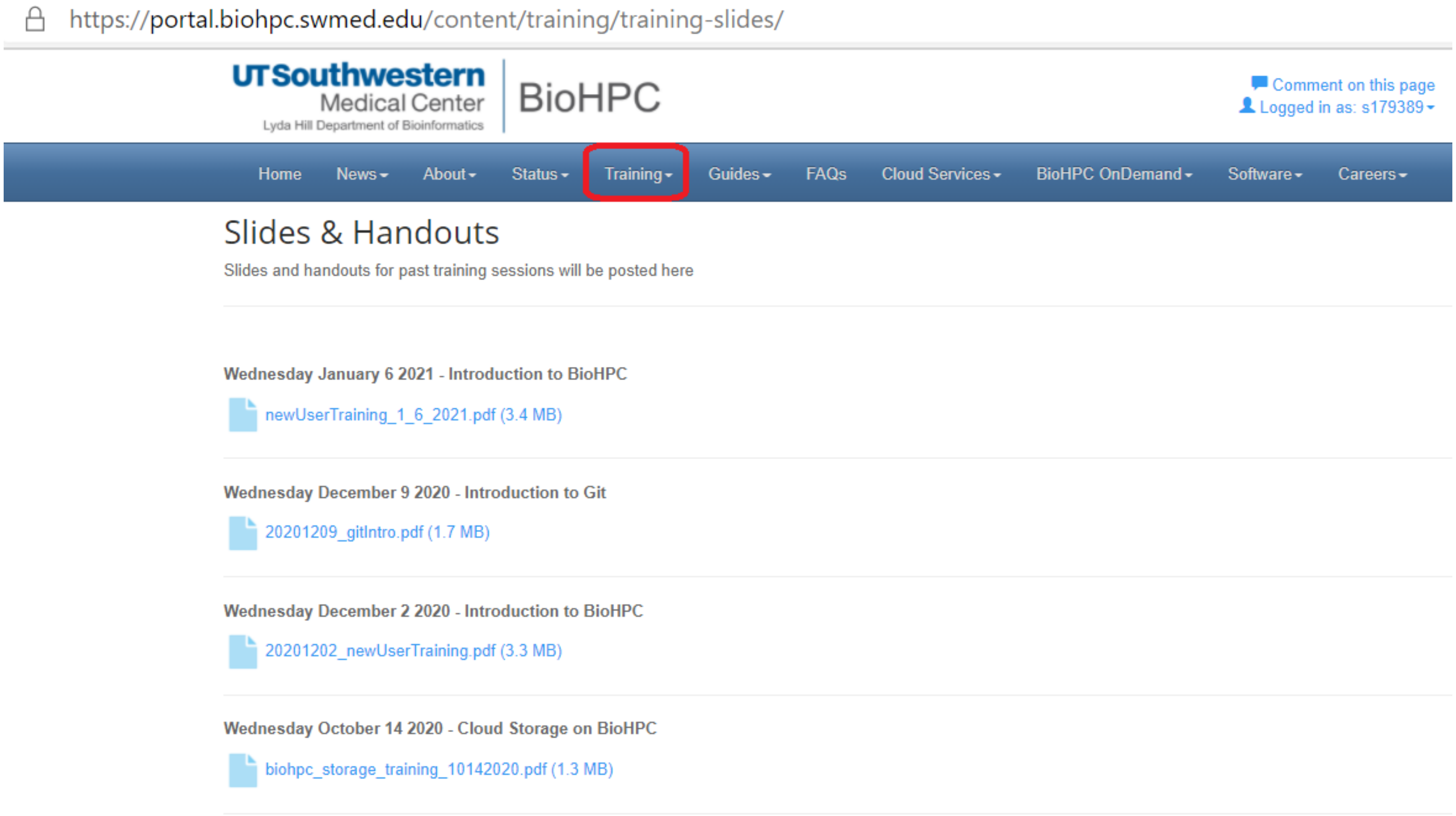
Multiprocessing - Shared Data using a Manager multiproc_manager_2021.py



```python
from multiprocessing import Manager, Pool
import os

def f(l, d):
    l.append('worker')
    d[str(os.getpid())] = 'worker'
manager = Manager()
pool = Pool(2)

# private_l and private_d only visible to local process
private_l = list()
private_d = dict()

# shared_l and shared_d visible to every process
shared_l = manager.list()
shared_d = manager.dict()

# manager process can see this change
private_l.append('manager')
private_d[str(os.getpid())] = 'manager'

# manager process can see this change
shared_l.append('manager')
shared_d[str(os.getpid())] = 'manager'

# changes child processes makes are lost
pool.apply(f, args=(private_l, private_d))
pool.apply(f, args=(private_l, private_d))
print ("try to add to private data", private_l, private_d)

# changes child processes makes are kept
pool.apply(f, args=(shared_l, shared_d))
pool.apply(f, args=(shared_l, shared_d))
print ("try to add to shared data", shared_l, shared_d)
```

```
Output[]:
try to add to private data ['manager'] {'56636':
'manager'}
try to add to shared data ['manager', 'worker',
'worker'] {'56636': 'manager', '58800': 'worker',
'58802': 'worker'}
```

Codes availibility



https://portal.biohpc.swmed.edu/content/training/training-slides/

Slides & Handouts

Slides and handouts for past training sessions will be posted here

Wednesday January 6 2021 - Introduction to BioHPC

newUserTraining_1_6_2021.pdf (3.4 MB)

Wednesday December 9 2020 - Introduction to Git

20201209_gitIntro.pdf (1.7 MB)

Wednesday December 2 2020 - Introduction to BioHPC

20201202_newUserTraining.pdf (3.3 MB)

Wednesday October 14 2020 - Cloud Storage on BioHPC

biohpc_storage_training_10142020.pdf (1.3 MB)

https://portal.biohpc.swmed.edu/content/training/training-slides/

UT Southwestern Medical Center
Lyda Hill Department of Bioinformatics

BioHPC

Thanks!