Python Usage Metrics on Blue Waters

Colin A. MacLean
National Center for Supercomputing Applications
University of Illinois
Urbana, Illinois
Email: cmaclean@illinois.edu

***Abstract*—Blue Waters supports a large Python stack containing over 650 total packages. As part of maintaining this support, logging functionality has been introduced to track the usage statistics of both National Center for Supercomputing Applications (NCSA) and user provided Python packages. Due to the number of NCSA supplied packages, it is rare to receive a request for packages which are not already installed, leading to a lack of information about which packages and their dependencies are being used. By tracking module imports, a detailed log of usage information has been used to focus support efforts on improving the usability and performance of popular usage patterns.**

***Index Terms*—Python; Cray; HPC; Blue Waters;**

I. Introduction

Python is a programming language which has seen rapid growth in scientific computing. Despite Python itself being poorly suited to parallel programming due to its Global Interpreter Lock (GIL), Python has been increasingly used[3] for the rapid development of scientific software. Python excels as a high level glue to interface high performance native libraries such as Numpy, Scipy, and Tensorflow. New libraries are increasingly being written for use with Python, if not as the primary interface, then at least with Python bindings. This is particularly the case in the field of machine learning[1].

II. Python on Blue Waters

Python on Blue Waters, BWPY, is built and maintained by a patched installation of Gentoo Portage[4]. The goal of BWPY is to provide users with a large number of preinstalled packages built against optimized libraries. Because Blue Waters uses the aging SLES 11 GNU/Linux as its operating system, the ability for Portage to manage many non-Python dependencies as well as Python packages is important for ensuring that BWPY provides access to up-to-date releases of software packages. Python packages in BWPY are currently built for CPython 2.7, 3.3, 3.4, 3.5, PyPy, and PyPy 3, depending on package compatibility. For the Python 3 branch, Python 3.4 is the default Python 3. Python 2.7 is the default Python. Patches to Gentoo Portage also provide functionality for creating a hierarchy of environment modules, which are used to customize the Python environment.

BWPY currently tracks 661 packages. Approximately 230 of these are Python packages, the rest being libraries which would otherwise be too old or unavailable. The BWPY-MPI sub-module overlays BWPY with 13 packages which are either rebuilt with MPI support or require MPI. Because MPI processes cannot run on login nodes or MOM nodes, splitting this functionality from the main BWPY collection is necessary for ensuring Python is usable in all environments users may encounter.

Tensorflow is also provided via a sub-module. This is in response to Tensorflow being difficult to build on systems with nonstandard toolchains and library locations and multiple users requesting frequent updates to this rapidly developing machine learning framework. Installing Tensorflow as a sub-module allows for multiple versions to be available and prevents frequent changes to the main BWPY prefix. The modularization also prevents the introduction of unstable dependencies into the main BWPY prefix which may be used by other packages.

III. Instrumentation

The Python module imports were logged via a `sitecustomize.py` exit hook. This exit hook uses `sys.modules` to list the module name and the file location of the modules imported at the end of Python's execution. Additionally, the username, timestamp, execution context (login node, MOM node, or compute node), number of nodes and processes per node, and sub-module usage were recorded. This data was written to plain text log files. Because all users needed write access to these files, the logs were given the +a (append only) file attribute to prevent accidental deletion or truncation.

The advantage of this logging method, as opposed to intercepting import calls, is that any errors in `sitecustomize.py` are non-fatal. This allowed for

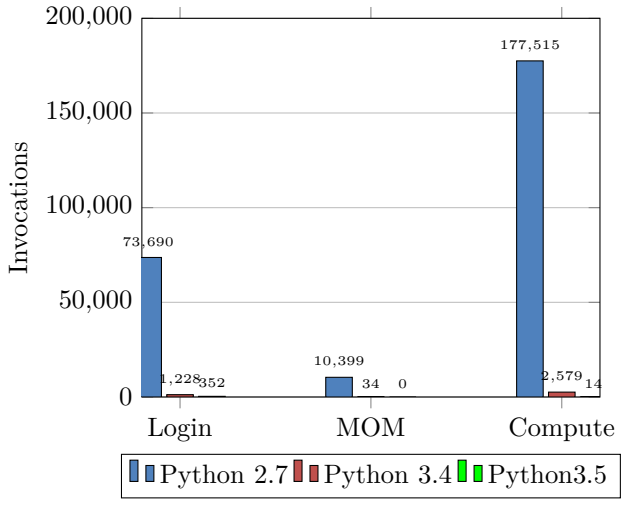
tweaking the logging without the worry of impacting users or any performance penalty. The disadvantage of this method is that not all imports are visible in the global `sys.modules` upon program termination, such as Python scripts invoked via `exec` and Virtualenv containers not inheriting system site-packages. This logging does not record invocations which crash or are killed.

IV. Results

Between January 13th, 2017 and April 2nd, 2017, Python 2.7 was used by 126 users. Python 3.3, an old version kept in case of possible compatibility problems, was not used at all. Python 3.4 was used by 28 users and Python 3.5 was used by 8 users. Python 2.7 was invoked 264,064 times, Python 3.4 was invoked 4828 times, and Python 3.5 was invoked 775 times. Despite Python 3 having been released over 8 years ago, Python 2 remains by far the most dominant branch of Python used in HPC. The most likely reason Python 3 has seen low adoption is that many of the lower level libraries took a significant amount of time to be ported to Python 3 and some still remain incompatible with the Python 3.5 branch. PyPy and PyPy 3 have not seen any adoption by Blue Waters users. For a detailed table of Python modules used by multiple users on Blue Waters, see Appendix.

A. Python Versions and Node Usage

Fig. 1. Python Usage by Type (Invocations)



As seen in Fig. 1, Python is consistently used across login nodes, MOM nodes, and compute nodes. Based upon the packages used and the contexts in which Python is invoked, Python is used for data processing, analysis, and visualization, job processing, and directly for scientific computing purposes.

B. Fields of Science

Logged job IDs were used to associate jobs and Python users with their respective fields of science.

Fig. 2. Workloads of Python Users

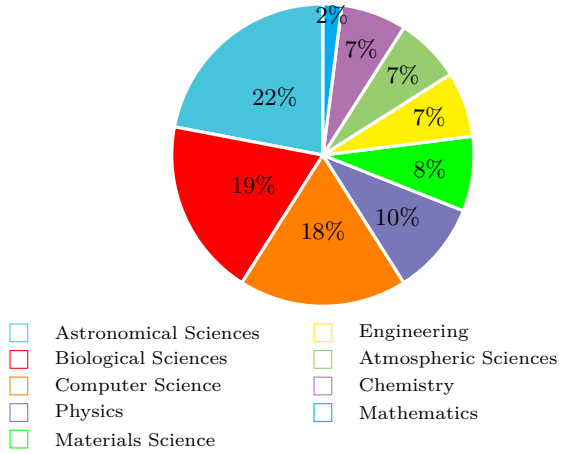


Fig. 3. Workloads of Python Invocations

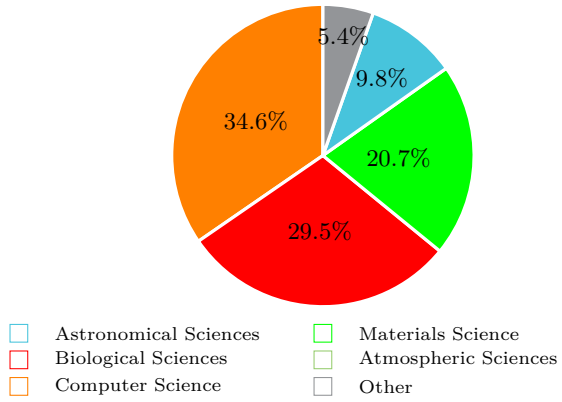


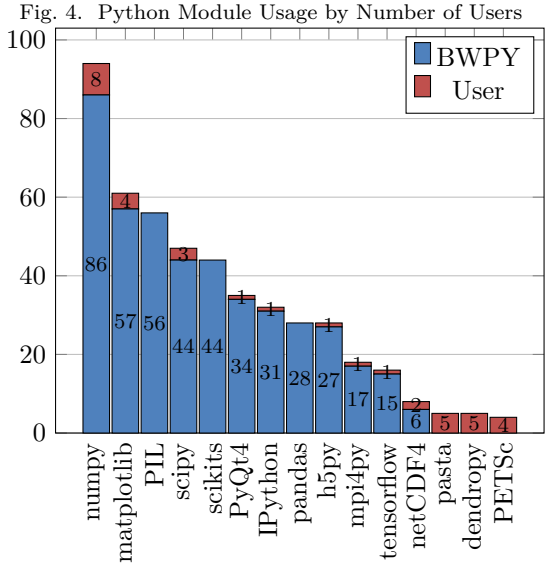
Fig. 2 shows that Python is used across diverse areas of science. However, it can also be seen from Fig. 3 that these areas of science tend to use Python differently. Atmospheric sciences account for the most users of Python on Blue Waters, but only account for <10% of Python invocations. This lack of correlation is likely due to Python being used in a more supportive role in some fields versus a primary programming language in others.

C. Module Usage by Number of Users

The following graph, Fig. 4, lists the most popular Python modules related to scientific computing, determined by how many unique users imported these modules. This graph includes data from all available

Python versions. Utility modules such as `six` are not included in this graph.

Fig. 4. Python Module Usage by Number of Users



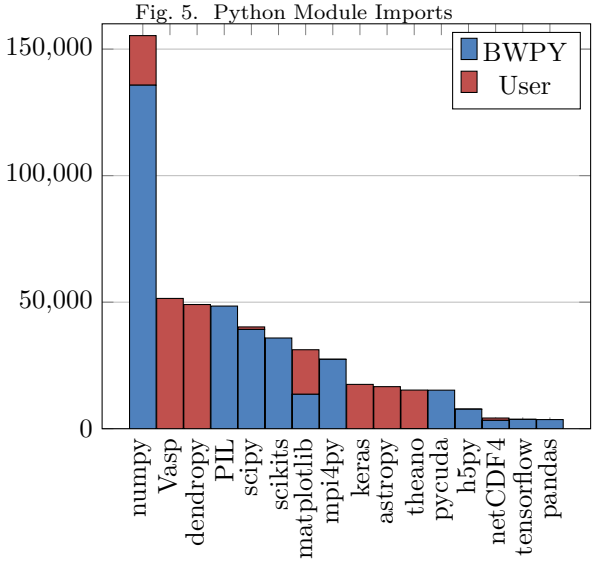
Numpy is the most popular module of the BWPY Python stack, being used by more than 60% of Blue Waters Python users. Most Numpy users used the module provided by BWPY, but 8 built their own version. At least one user overrode all the major Python packages. One potential reason for users building their own versions of modules which have already been provided by BWPY is that the `pip --upgrade` command will upgrade BWPY provided commands when the user does not necessarily want or need to provide their own build. One user who contacted Blue Waters support built their own Numpy to use the libsci_mp OpenMP enabled BLAS.

D. Most Imported Modules

Numpy was also by far the most imported module on Blue Waters, imported over 150,000 times over the course of the data collection, over double that of the next most imported module, Vasp.

It is clear from comparing the module usage by number of users (Fig. 4) to the module usage by number of imports (Fig. 5) that Python is used as both a primary and secondary tool. `Vasp` is only used by a single user, but is the second most invoked scientific Python module. `Dendropy` is only used by 5 users but is the 3rd most invoked module. `Matplotlib` is the second most popular module with 61 users but is only the 7th most invoked. This lack of correlation between the number of users and number of imports is because a minority of users use Python for performing primary scientific calculations, resulting in a high number of invocations, while a larger number of

Fig. 5. Python Module Imports



users employ Python less often in a supportive role for data analysis and visualization.

E. Python Job Walltimes

Using the logged job IDs, the walltimes of Compute and MOM node jobs were found from the job scheduler logs. It should be noted that Python could be run multiple times within a single job or Python may only be used briefly. The Python `atexit` logging could also record Python execution walltimes. As the focus of this logging was initially Python module usage, additional logging including Python walltime was not yet implemented. Python execution walltime could also be misleading if Python is merely used to execute other programs with Python sleeping as a parent process. Thus, it is not trivial to precisely determine the length of time Python processes are run.

The vast majority of Python jobs were under four hours of walltime. 10893 jobs ran for less than 15 minutes, 7004 jobs ran between 15 and 30 minutes, and 6288 jobs ran between one half to one hour. Only 162 jobs ran for more than 16 hours.

The high number of short jobs was noteworthy and concerning. Such usage causes extra work for the scheduler compared to bundling short jobs into longer jobs. Additionally, Python on Blue Waters is installed on a Lustre filesystem and there are a large amount of IOPs required to launch Python. Ideally, running Python on multiple datasets would be done within a single invocation of Python, reducing unnecessary load to the filesystem.

Fig. 6. Compute Node Job Walltimes

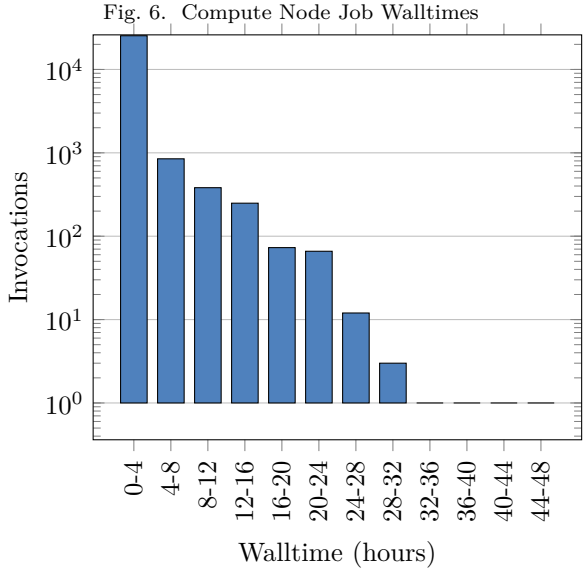


Fig. 7. MOM Node Job Walltimes

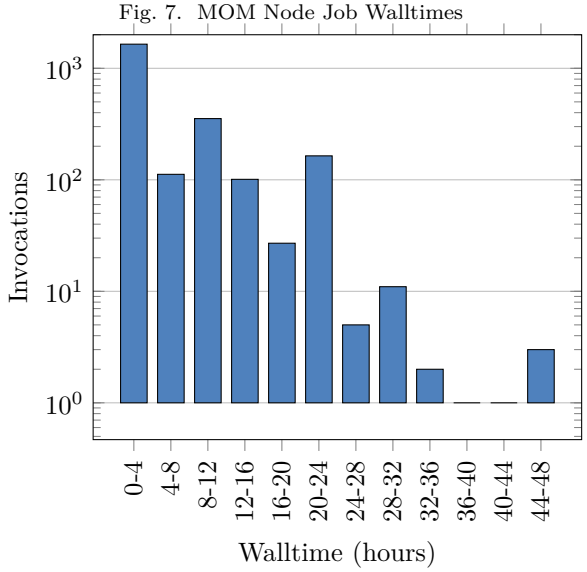


Fig. 8. Executions Per Job

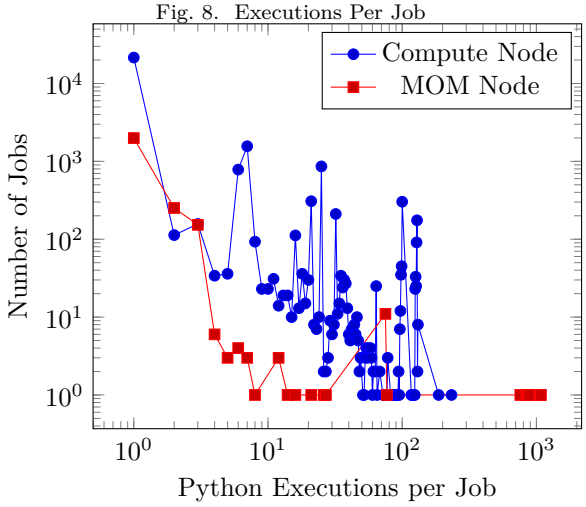
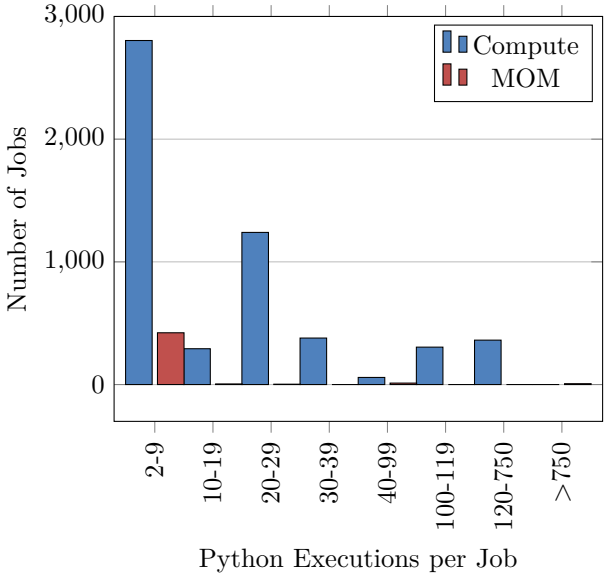


Fig. 9. Multiple Launch Breakdown



Python processes running on MOM nodes tended to run longer than on compute nodes. This suggests that Python is being used to bundle other programs for execution on compute nodes. It is also likely that some of these jobs mistakenly ran Python without using `aprun`.

F. Python Usage in Jobs

By counting duplicate job IDs in the log, the frequency of Python invocation within jobs was calculated. Fig. 8 and Fig. 9 show how often Python is executed per job.

Similarly to running many short Python jobs, running Python many times within the same job puts strain on the Lustre filesystem. When possible, users should be encouraged to limit the number of Python invocations by creating a Python script to bundle multiple tasks within the same invocation of Python.

G. Compute Job Width Distribution

The Python `atexit` logging also recorded the PBS job width and processes per node using the environment variables set by the scheduler. This data was collected for both compute node and MOM node jobs.

TABLE I
PYTHON COMPUTE JOB WIDTHS

# of nodes	Invocations
1	170488
2-32	8555
33-64	53
65-128	222
129-256	600
257-512	94
513-1024	71
1025-2048	25

As can be seen in Table I, the vast majority of Python compute node jobs were single node jobs. However, there were some jobs which used many nodes. Unlike invoking Python multiple times per job or running many short jobs, the load incurred by launching Python across multiple nodes cannot be reduced by bundling tasks. Better support for wide workoads will require running Python in a manner which is more friendly to high IOPs than off the Lustre filesystem.

V. Implications for Python on Blue Waters

The Python usage statistics indicate that the packages provided by BWPY more than sufficiently cover the needs of Blue Waters Python users. Rather than continuing to proactively expand the number of packages supported by BWPY, it was decided that it would be more beneficial to provide multiple options for the most commonly used packages.

Due to the high frequency of Numpy/Scipy usage, giving users multiple BLAS options was deemed to be the most useful expansion of BWPY. A user had also expressed interest in an OpenMP enabled BLAS with Numpy, which was additional evidence that multiple options would be appreciated by the users of Python on Blue Waters. Using the Gentoo utility `revdep-rebuild.sh`, a list of 10 packages were found to be linking against Cray's libsci. Two new submodules were added, providing alternative builds of these packages using `libsci_mp` and `libsci_acc` for OpenMP enabled BLAS and GPGPU enabled BLAS.

Blue Waters users will be given advice for bundling their Python tasks to reduce startup cost. Providing BWPY as a mounted image to reduce Lustre metadata load could also significantly improve interactive experience and scalability; the use of mounted images in Shifter was shown to improve performance to that of ramdisks[2].

Despite Numpy being used by the majority of users, it is not used unanimously. Those users which need limited or no Numpy functionality should be encouraged to try running under PyPy or PyPy3. These Python implementations use JIT compilation to achieve better performance.

References

[1] Martín Abadi et al. "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems". In: (Mar. 2016). arXiv: 1603.04467. URL: http://download.tensorflow.org/paper/whitepaper2015.pdf%20http://arxiv.org/abs/1603.04467.

[2] Richard Shane Canon and Doug Jacobsen. "Shifter: Containers for HPC". In: (2017). URL: https://cug.org/proceedings/cug2016_proceedings/includes/files/pap103.pdf.

[3] Matthew D Jones et al. *FINAL REPORT: WORKLOAD ANALYSIS OF BLUE WATERS*. Tech. rep. ACI, 2017, p. 25. URL: https://arxiv.org/pdf/1703.00924.pdf.

[4] Colin A. MacLean. "Maintaining Large Software Stacks in a Cray Ecosystem with Gentoo Portage". In: *Cray User Group* (2016).

Appendix

Module	Invocations BWPY	Invocations User	Users BWPY	Users User
CIME	0	1858	0	2
Cookie	49	0	3	0
Cython	330	1	20	1
FixTk	15379	0	3	0
HTMLParser	39	0	3	0
IPython	8663	64	31	1
OpenSSL	885	0	36	0
PETSc	0	24	0	4
PIL	48459	0	56	0
PyQt4	1719	7	34	1
Queue	15407	0	3	0
RDict	0	24	0	4
SCons	1	24	1	2
SNP_class	0	764	0	2
StringIO	15486	0	3	0
Tools	0	128	0	2
apitools	0	24	0	2
appdirs	191	1362	4	6
archive	0	688	0	3
argcomplete	12	0	2	0
argparse	149	20	3	4
args	0	24	0	4
arpack	0	8	0	2
ast	15047	0	4	0
astropy	0	16619	0	2
atari_py	0	245	0	2
atexit	15643	0	3	0
babel	27	0	5	0
backports	14593	64	45	1
base64	15458	0	3	0
bisect	15289	0	3	0
blopex	0	8	0	2
blzpack	0	8	0	2
boto	3880	12	28	2
bottleneck	4002	0	28	0
bson	3689	0	2	0
btmorph	0	90	0	2
build	0	20	0	3
buildnml	0	70	0	2
builtins	0	13910	0	2
cairo	168	0	3	0
calendar	15462	0	4	0
certifi	522	0	39	0
cffi	47824	819	56	1
cgi	53	0	4	0
chardet	545	0	14	0
check	0	8	0	2
cloudpickle	3287	0	13	0
cmakeboot	0	20	0	4
cmakegen	0	43	0	4
collections	15525	0	3	0
colorama	1197	0	7	0
common	0	37	0	2
compileall	28	0	4	0
concurrent	7935	0	13	0
config	0	224	0	5
contextlib	15493	0	4	0
cookielib	49	0	3	0
copy	15497	0	3	0
coverage	18	0	4	0
crcmod	0	12	0	2
csv	15278	0	4	0
ctypes	15480	0	4	0
cutils_ext	44	2	4	1
cv2	3	61	2	2
cycler	29972	1179	60	2
cython	49	1	6	1
dask	3288	0	13	0
dateutil	13603	17621	59	6
decimal	15251	0	4	0
decorator	11022	65	32	1
dendropy	0	49069	0	5
difflib	15432	0	4	0
dis	15430	0	4	0
docutils	27	0	5	0
email	15536	0	4	0
enum	456	70	34	3
ez_setup	0	11	0	5
feast	0	8	0	2
fractions	15251	0	4	0
funcsigs	255	0	10	0
functools	15487	0	3	0
functools32	16428	0	7	0
future	0	14105	0	2
getopt	15541	0	4	0
getpass	28	0	4	0
gettext	179	0	4	0
gflags	0	12	0	2
glob	15652	0	4	0
globus_sdk	6	4	1	1
gmpy2	1580	0	5	0
google	19	43	3	3
graph	0	24	0	4

	Invocations		Users			Invocations		Users	
Module	**BWPY**	**User**	**BWPY**	**User**	**Module**	**BWPY**	**User**	**BWPY**	**User**
gsd	0	3695	0	2	nbformat	92	0	6	0
gslib	0	12	0	2	ndg	64	0	8	0
gsutil	0	12	0	2	netCDF4	3306	918	7	2
gym	8	307	1	2	netcdftime	3306	918	7	2
gzip	15411	0	4	0	netifaces	156	0	5	0
h5py	7701	2	27	1	netrc	8	0	4	0
hashlib	15487	0	3	0	neuron	0	877	0	2
healpy	0	16431	0	2	neurotrees	0	987	0	2
heapq	15525	0	3	0	nose	251	0	8	0
help	0	24	0	4	notebook	91	0	6	0
hmac	52	0	3	0	numbers	15454	0	4	0
hoomd	15	143	2	3	numexpr	3890	0	28	0
html	8	10	1	1	numpy	135741	19591	93	8
http	10	14	1	1	opcode	15519	0	4	0
httplib	72	0	3	0	optparse	30	0	4	0
httplib2	0	24	0	2	osgeo	0	4	0	2
idna	886	0	36	0	overview	0	11	0	2
importlib	15472	0	3	0	packaging	0	1553	0	10
inspect	15430	0	4	0	pandas	3878	0	28	0
io	15487	0	3	0	parfile	0	180	0	3
ipaddress	487	1	36	1	pasta	0	3030	0	5
ipykernel	8137	0	14	0	pathlib	22138	0	49	0
ipyparallel	133	0	5	0	pbr	551	12	31	1
ipywidgets	407	0	12	0	petscconf	0	8	0	2
jinja2	119	0	10	0	pexpect	8671	64	31	1
joblib	393	0	14	0	pickle	15444	0	4	0
json	15314	0	4	0	pickleshare	8572	64	31	1
jsonschema	92	7	6	1	pip	319	52	29	9
keras	0	17513	0	6	pkgutil	15461	0	4	0
keyword	15525	0	3	0	platform	15652	0	4	0
lapack	0	8	0	2	plistlib	15461	0	4	0
libutil	0	688	0	3	polp_new	0	30	0	2
llnl	0	8	0	2	pprint	15492	0	4	0
log	0	8	0	2	primme	0	8	0	2
logger	0	24	0	4	project	0	20	0	3
logging	15455	0	4	0	psutil	3324	0	16	0
lxml	20	0	3	0	ptyprocess	8671	64	31	1
markupbase	39	0	3	0	py	3884	7	29	1
markupsafe	119	0	10	0	py_compile	70	0	4	0
matplotlib	13596	17609	59	4	pyasn1	870	24	35	2
mimetools	72	0	3	0	pycuda	15230	0	2	0
mimetypes	56	0	4	0	pydot	72	15217	5	1
miscVar	0	339	0	2	pygments	8718	64	35	1
mock	189	0	7	0	pyini	0	688	0	3
mpdlib	0	7	0	2	pylab	7014	36	9	2
mpi4py	27410	2	17	1	pynbody	0	24	0	2
mpmath	460	1120	5	1	pyparsing	28342	19440	58	13
nargs	0	24	0	4	pytest	0	10	0	2

	Invocations		Users	
Module	**BWPY**	**User**	**BWPY**	**User**
pytz	3912	0	32	0
pyu2f	0	12	0	2
qhull	27	0	3	0
queue	4	7408	1	1
quicklens	0	4660	0	2
quopri	15462	0	4	0
random	15487	4	3	1
redis	374	0	32	0
requests	534	0	14	0
restartlib	0	688	0	3
retrieval	0	17	0	4
rfc822	72	0	3	0
rlcompleter	33	0	3	0
rsa	0	24	0	2
scikits	35843	0	44	0
scipy	39232	986	47	3
script	0	24	0	4
seaborn	187	0	6	0
sepp	0	1182	0	2
setup	0	12	0	2
setuptools	612	57	27	9
shlex	15262	0	4	0
shutil	15487	0	3	0
sim-build	0	688	0	3
sim-info	0	688	0	3
sim-manage	0	688	0	3
sim-sync	0	688	0	3
sim-util	0	688	0	3
simarchive	0	688	0	3
simdb	0	688	0	3
simdt	0	688	0	3
simenv	0	688	0	3
simlib	0	688	0	3
simopts	0	688	0	3
simplejson	12458	0	51	0
simremote	0	688	0	3
simrestart	0	688	0	3
simsubs	0	688	0	3
simtk	73	151	2	1
sip	1715	7	35	1
six	52177	1409	89	8
sklearn	106	13	6	1
socket	15463	0	4	0
socks	330	24	31	2
sphinx	27	0	5	0
ssl	15463	0	4	0
statsmodels	188	0	6	0
storemagic	214	14	19	1
string	15649	0	4	0
stringprep	13	0	4	0
struct	15487	0	3	0
subprocess	15447	0	4	0
sympy	460	1120	5	1
sysconfig	15461	0	4	0
tables	44	0	2	0
tarfile	67	0	3	0
tempfile	15487	0	3	0
tensorflow	3676	26	16	1
terminado	87	0	6	0
tests	0	12	0	2
textwrap	15464	0	4	0
theano	24	15242	4	2
threading	15495	0	4	0
timeit	28	0	2	0
token	15445	0	3	0
tokenize	15445	0	3	0
toolz	3294	0	13	0
tornado	8139	0	14	0
tqdm	0	52	0	3
traceback	15519	0	4	0
traitlets	8677	64	32	1
trlan	0	8	0	2
unittest	15434	0	4	0
urllib	15470	0	4	0
urllib2	72	0	3	0
urllib3	667	0	18	0
urlparse	15458	0	3	0
util	0	73	0	2
utils	0	44	0	4
uu	15462	0	4	0
uuid	56	0	4	0
version	2441	22	1	3
vgg_model	0	31	0	2
virtualenv	16	4	7	2
wcwidth	8668	64	32	1
weakref	15504	0	3	0
webcolors	81	0	8	0
wheel	379	29	31	5
xml	233	0	4	0
xmlrpclib	24	0	3	0
yaml	917	8	7	2
yt	50	1119	2	1
zipfile	15461	0	4	0
zmq	8179	0	15	0