ch01-intro

April 29, 2021

0.0.1 Chapter 1: Computing with Python

0.0.2 Overview: a typical Python-based scientific computing stack.

0.0.3 Resources:

- Intel MKL (Math Kernel Library)
- openBLAS
- ATLAS
- SciPy
- Python Numeric & Scientific topics

0.0.4 Interpreter

- The easist way to execute Python code: run the program directly.
- Use Jupyter magic command to write Python source file to disk:

[1]:
```
%%writefile hello.py
print("Hello from Python!")
```

```
Overwriting hello.py
```

- Use the ! system shell command (included in the Python Jupyter kernel) to interactively run Python with hello.py as its argument.

[2]:
```
!python hello.py
```

```
Hello from Python!
```

[3]:
```
!python --version
```

```
Python 3.8.2
```

0.0.5 Input and output caching

- Input & output history can be accessed using **In** (a list) & **Out** (a dictionary). Both can be indexed with a cell number.

[4]:
```
3*3
In[1]
```

[4]:
```
'get_ipython().run_cell_magic(\'writefile\', \'hello.py\', \'print("Hello from
Python!")\\n\')'
```

- A single underscore = the most recent output;
- A double underscore = the *next* most recent output.

[5]:
```
1+1
```

[5]:
```
2
```

[6]:
```
2+2
```

[6]:
```
4
```

[7]:
```
_, __
```

[7]:
```
(4, 2)
```

[8]:
```
# In = a list
In
```

[8]:
```
['',
 'get_ipython().run_cell_magic(\'writefile\', \'hello.py\', \'print("Hello from
Python!")\\n\')',
 "get_ipython().system('python hello.py')",
```

```
 "get_ipython().system('python --version')",
 '3*3\nIn[1]',
 '1+1',
 '2+2',
 '_, __',
 '# In = a list\nIn']
```

[9]:
```
# Out = a dictionary
Out
```

[9]:
```
{4: 'get_ipython().run_cell_magic(\'writefile\', \'hello.py\', \'print("Hello
from Python!")\\n\')',
 5: 2,
 6: 4,
 7: (4, 2),
 8: ['',
  'get_ipython().run_cell_magic(\'writefile\', \'hello.py\', \'print("Hello from
Python!")\\n\')',
  "get_ipython().system('python hello.py')",
  "get_ipython().system('python --version')",
  '3*3\nIn[1]',
  '1+1',
  '2+2',
  '_, __',
  '# In = a list\nIn',
  '# Out = a dictionary\nOut']}
```

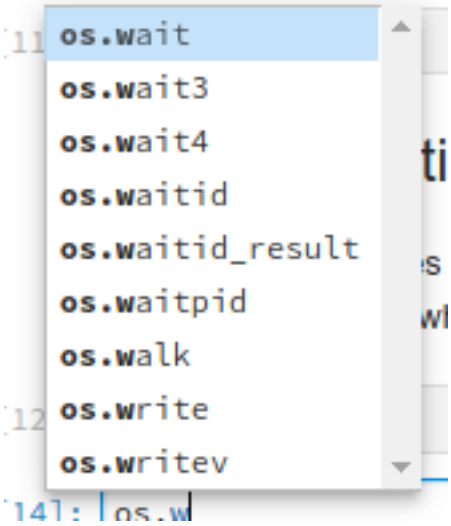
[10]:
```
# Suppress output results by ending statement with a semicolon
1+2;
```

0.0.6 Autocompletion

- The **Tab** key activates autocompletion (displays list of symbol names that are valid completions of what has been typed thus far.)

[11]:
```
import os
```

- Results of typing "os.w", followed by ?



0.0.7 Documentation

- "Docstrings" provide a built-in reference manual for most Python modules. Display the docstring by appending a Python object with "?".

[12]:
```
import math
math.cos?
```

```
Signature: math.cos(x, /)
Docstring: Return the cosine of x (measured in radians).
Type:      builtin_function_or_method
```

0.0.8 Interaction with System Shell

- Anything after ! is evaluated using the system shell, such as bash.
- (I use Ubuntu Linux as my laptop OS. Your Windows equivalents will vary.)

[13]:
```
!touch file1.py file2.py file3.py
!ls file*
```

```
file1.py  file2.py  file3.py
```

[14]:
```
# output of a system shell command can be captured in a Python variable
files = !ls file*
print(len(files))
print(files)
```

```
3
['file1.py', 'file2.py', 'file3.py']
```

[15]:
```
# pass Python variable values to shell commands
# by prefixing the variable name with $.
file = "file1.py"
!ls -l $file
```

```
-rw-rw-r-- 1 bjpcjp bjpcjp 0 Apr 29 06:19 file1.py
```

0.0.9 IPython Extensions

- Commands start with one or two "%" characters. A single % is used for single-line commands; dual %% is used for cells (multiple lines).
- %lsmagic returns a list of available commands.

[16]:
```
%lsmagic
```

[16]:
```
Available line magics:
%alias  %alias_magic  %autoawait  %autocall  %automagic  %autosave  %bookmark
%cat  %cd  %clear  %colors  %conda  %config  %connect_info  %cp  %debug  %dhist
%dirs  %doctest_mode  %ed  %edit  %env  %gui  %hist  %history  %killbgscripts
%ldir  %less  %lf  %lk  %ll  %load  %load_ext  %loadpy  %logoff  %logon
%logstart  %logstate  %logstop  %ls  %lsmagic  %lx  %macro  %magic  %man
%matplotlib  %mkdir  %more  %mv  %notebook  %page  %pastebin  %pdb  %pdef  %pdoc
%pfile  %pinfo  %pinfo2  %pip  %popd  %pprint  %precision  %prun  %psearch
%psource  %pushd  %pwd  %pycat  %pylab  %qtconsole  %quickref  %recall  %rehashx
%reload_ext  %rep  %rerun  %reset  %reset_selective  %rm  %rmdir  %run  %save
%sc  %set_env  %store  %sx  %system  %tb  %time  %timeit  %unalias  %unload_ext
%who  %who_ls  %whos  %xdel  %xmode

Available cell magics:
%%!  %%HTML  %%SVG  %%bash  %%capture  %%debug  %%file  %%html  %%javascript
%%js  %%latex  %%markdown  %%perl  %%prun  %%pypy  %%python  %%python2
%%python3  %%ruby  %%script  %%sh  %%svg  %%sx  %%system  %%time  %%timeit
%%writefile

Automagic is ON, % prefix IS NOT needed for line magics.
```

0.0.10 Running scripts

- %run executes an external Python source file within an interactive IPython session.

[17]:
```
%%writefile fib.py

def fib(N):
    """
    Return a list of the first N Fibonacci numbers.
    """
    f0, f1 = 0, 1
```

```
    f = [1] * N
    for n in range(1, N):
        f[n] = f0 + f1
        f0, f1 = f1, f[n]

    return f

print(fib(10))
```

```
Overwriting fib.py
```

[18]:
```
!python fib.py
```

```
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
```

[19]:
```
%run fib.py
```

```
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
```

[20]:
```
fib(6)
```

[20]:
```
[1, 1, 2, 3, 5, 8]
```

0.1 Listing all defined symbols

- %who lists all defined symbols
- %whos provides more detailed info.

[21]:
```
%who
```

```
fib      file    files   math    os
```

[22]:
```
%whos
```

```
Variable   Type        Data/Info
-------------------------------
fib        function    <function fib at 0x7fdfcafae1f0>
file       str         file1.py
files      SList       ['file1.py', 'file2.py', 'file3.py']
math       module      <module 'math' from '/hom<…>-38-x86_64-linux-gnu.so'>
os         module      <module 'os' from '/home/<…>ing/lib/python3.8/os.py'>
```

0.2 Debugger

- Use %debug to step directly into the Python debugger.

[23]:
```
# fib function fails - can't use floating point numbers.
try:
    fib(1.0)
```

```
except TypeError:
    print("nope. can't do that.")
```

```
nope. can't do that.
```

[24]:
```
#%debug
```

0.2.1 Resetting the Python namespace

- Ensures a program is run in a pristine environment, uncluttered by existing variables and functions. Although it is necessary to reimport modules, it is important to know that even if the modules have changed since the last import, a new import after a %reset will not import the new module but rather reenable a cached version from the previous import.
- When developing Python modules, this is usually not the desired behavior. In that case, a reimport of a previously imported (and since updated) module can often be achieved by using the reload function from `IPython.lib.deepreload`. However, this method does not always work, as some libraries run code at import time that is only intended to run once. In this case, the only option might be to terminate and restart the IPython interpreter.

[26]:
```
%reset
```

```
Once deleted, variables cannot be recovered. Proceed (y/[n])?  n

Nothing done.
```

0.3 Timing and profiling code

- `%timeit` and `%time` provide simple benchmarking utilities.

[25]:
```
# first, re-define fibonacci code used above.
def fib(N):
    """
    Return a list of the first N Fibonacci numbers.
    """
    f0, f1 = 0, 1
    f = [1] * N
    for n in range(1, N):
        f[n] = f0 + f1
        f0, f1 = f1, f[n]

    return f
```

[26]:
```
# timeit does not return expression's resulting value.
%timeit fib(50)
```

```
3.11 µs ± 32.5 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

[27]:
```
# %time only runs once. less accurate estimate.
result = %time fib(100)
```

```
CPU times: user 12 µs, sys: 0 ns, total: 12 µs
Wall time: 13.6 µs
```

[28]:
```
len(result)
```

[28]:
```
100
```

- The **cProfile** module provides **%prun** (for statements) and **%run** (for external scripts) profiling commands.

[29]:
```
import numpy as np

def random_walker_max_distance(M, N):
    """
    Simulate N random walkers taking M steps
    Return the largest distance from the starting point.
    """
    trajectories = [np.random.randn(M).cumsum()
                    for _ in range(N)]
    return np.max(np.abs(trajectories))
```

[30]:
```
# returns call counts, runtime & cume runtime for
# each function.
%prun random_walker_max_distance(400, 10000)
```

```
         20013 function calls in 0.137 seconds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
    10000    0.089    0.000    0.089    0.000 {method 'randn' of 'numpy.random.
↪mtrand.RandomState' objects}
    10000    0.027    0.000    0.027    0.000 {method 'cumsum' of 'numpy.
↪ndarray' objects}
        1    0.013    0.013    0.135    0.135 <ipython-input-29-8a67fb24c99e>:
↪3(random_walker_max_distance)
        1    0.005    0.005    0.121    0.121 <ipython-input-29-8a67fb24c99e>:
↪8(<listcomp>)
        1    0.002    0.002    0.137    0.137 <string>:1(<module>)
        1    0.002    0.002    0.002    0.002 {method 'reduce' of 'numpy.ufunc'␣
↪objects}
        1    0.000    0.000    0.137    0.137 {built-in method builtins.exec}
        1    0.000    0.000    0.002    0.002 fromnumeric.py:70(_wrapreduction)
```

```
        1    0.000    0.000    0.002    0.002 <__array_function__ internals>:
↪2(amax)
        1    0.000    0.000    0.002    0.002 fromnumeric.py:2589(amax)
        1    0.000    0.000    0.002    0.002 {built-in method numpy.core.
↪_multiarray_umath.implement_array_function}
        1    0.000    0.000    0.000    0.000 fromnumeric.py:71(<dictcomp>)
        1    0.000    0.000    0.000    0.000 fromnumeric.py:
↪2584(_amax_dispatcher)
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.
↪Profiler' objects}
        1    0.000    0.000    0.000    0.000 {method 'items' of 'dict' objects}
```

0.3.1 Jupyter: External image rendering

[31]:
```
from IPython.display import display, Image, HTML, Math
```

[32]:
```
Image(url='http://python.org/images/python-logo.gif')
```

[32]:
```
<IPython.core.display.Image object>
```

0.3.2 Jupyter: HTML rendering

[33]:
```
import scipy, numpy, matplotlib
modules = [numpy, matplotlib, scipy]

row = "<tr><td>%s</td><td>%s</td></tr>"
rows = "\n".join(
    [row %
     (module.__name__, module.__version__)
     for module in modules])
table = "<table><tr><th>Library</th><th>Version</th></tr> %s </table>" % rows
```

[34]:
```
HTML(table)
```

[34]:
```
<IPython.core.display.HTML object>
```

[35]:
```
# another method
class HTMLdisplayer(object):
    def __init__(self,code):
        self.code = code
    def _repr_html_(self):
        return self.code

HTMLdisplayer(table)
```

[35]:
```
<__main__.HTMLdisplayer at 0x7fdfbc0c4970>
```

0.3.3 Jupyter: Formula rendering using Latex

[36]:
```
Math(r'\hat{H} = -\frac{1}{2}\epsilon \hat{\sigma}_z-\frac{1}{2}\delta␣
↪\hat{\sigma}_x')
```

[36]:
$$\hat{H} = -\frac{1}{2}\epsilon\hat{\sigma}_z - \frac{1}{2}\delta\hat{\sigma}_x$$

0.3.4 Jupyter: UI Widgets

** Needs debugging: slider widget doesn't appear. **

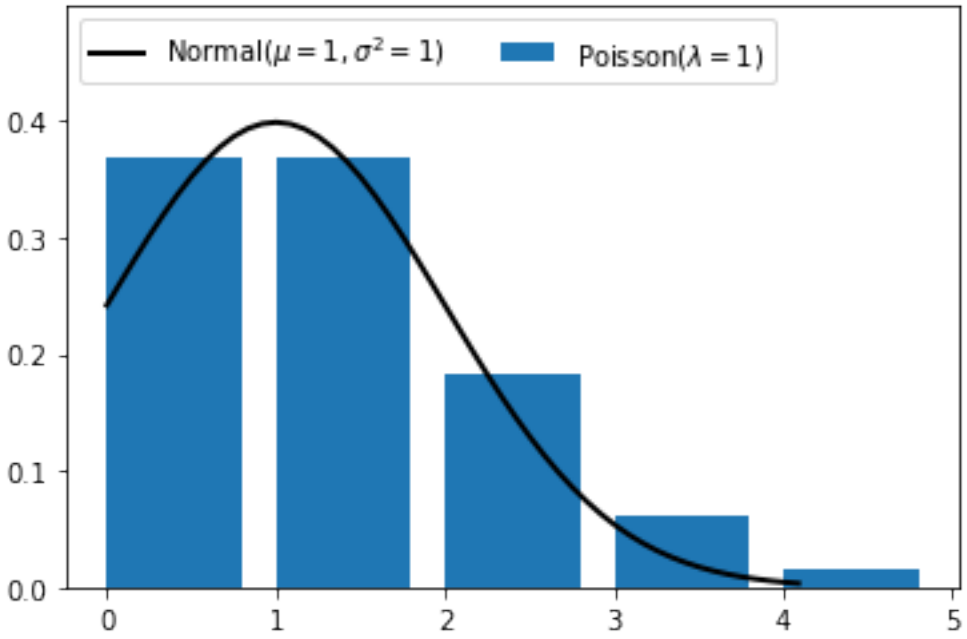
[37]:
```
import matplotlib.pyplot as plt
import numpy as np
from scipy import stats

def f(mu):
    X = stats.norm(loc=mu, scale=np.sqrt(mu))
    N = stats.poisson(mu)
    x = np.linspace(0, X.ppf(0.999))
    n = np.arange(0, x[-1])

    fig, ax = plt.subplots()
    ax.plot(x, X.pdf(x), color='black', lw=2, label="Normal($\mu=%d,␣
↪\sigma^2=%d$)" % (mu,mu))
    ax.bar(n, N.pmf(n), align='edge', label=r"Poisson($\lambda=%d$)" % mu)
    ax.set_ylim(0, X.pdf(x).max() * 1.25)
    ax.legend(loc=2, ncol=2)
    plt.close(fig)
    return fig
```

[38]:
```
from ipywidgets import interact
import ipywidgets as widgets
```

[39]:
```
interact(f, mu=widgets.FloatSlider(min=1.0, max=20.0, step=1.0));
```



0.3.5 nbconvert to HTML

[40]:
```
!jupyter nbconvert --to html ch01-intro.ipynb
```

```
[NbConvertApp] WARNING | Config option `kernel_spec_manager_class` not
recognized by `NbConvertApp`.
[NbConvertApp] Converting notebook ch01-intro.ipynb to html
[NbConvertApp] Writing 650798 bytes to ch01-intro.html
```

0.3.6 nbconvert to PDF

- Requires a LaTeX environment to be installed.
- On this system (Ubuntu Linux): `sudo apt-get install texlive-xetex`

[41]:
```
!jupyter nbconvert --to pdf ch01-intro.ipynb;
```

```
[NbConvertApp] WARNING | Config option `kernel_spec_manager_class` not
recognized by `NbConvertApp`.
[NbConvertApp] Converting notebook ch01-intro.ipynb to pdf
[NbConvertApp] Support files will be in ch01-intro_files/
[NbConvertApp] Making directory ./ch01-intro_files
[NbConvertApp] Writing 61529 bytes to notebook.tex
[NbConvertApp] Building PDF
Traceback (most recent call last):
  File "/home/bjpcjp/anaconda3/envs/working/bin/jupyter-nbconvert", line 11, in
<module>
```

```
    sys.exit(main())
  File "/home/bjpcjp/.local/lib/python3.8/site-
packages/jupyter_core/application.py", line 270, in launch_instance
    return super(JupyterApp, cls).launch_instance(argv=argv, **kwargs)
  File "/home/bjpcjp/.local/lib/python3.8/site-
packages/traitlets/config/application.py", line 845, in launch_instance
    app.start()
  File "/home/bjpcjp/.local/lib/python3.8/site-
packages/nbconvert/nbconvertapp.py", line 350, in start
    self.convert_notebooks()
  File "/home/bjpcjp/.local/lib/python3.8/site-
packages/nbconvert/nbconvertapp.py", line 524, in convert_notebooks
    self.convert_single_notebook(notebook_filename)
  File "/home/bjpcjp/.local/lib/python3.8/site-
packages/nbconvert/nbconvertapp.py", line 489, in convert_single_notebook
    output, resources = self.export_single_notebook(notebook_filename,
resources, input_buffer=input_buffer)
  File "/home/bjpcjp/.local/lib/python3.8/site-
packages/nbconvert/nbconvertapp.py", line 418, in export_single_notebook
    output, resources = self.exporter.from_filename(notebook_filename,
resources=resources)
  File "/home/bjpcjp/.local/lib/python3.8/site-
packages/nbconvert/exporters/exporter.py", line 181, in from_filename
    return self.from_file(f, resources=resources, **kw)
  File "/home/bjpcjp/.local/lib/python3.8/site-
packages/nbconvert/exporters/exporter.py", line 199, in from_file
    return self.from_notebook_node(nbformat.read(file_stream, as_version=4),
resources=resources, **kw)
  File "/home/bjpcjp/.local/lib/python3.8/site-
packages/nbconvert/exporters/pdf.py", line 183, in from_notebook_node
    self.run_latex(tex_file)
  File "/home/bjpcjp/.local/lib/python3.8/site-
packages/nbconvert/exporters/pdf.py", line 153, in run_latex
    return self.run_command(self.latex_command, filename,
  File "/home/bjpcjp/.local/lib/python3.8/site-
packages/nbconvert/exporters/pdf.py", line 110, in run_command
    raise OSError("{formatter} not found on PATH, if you have not installed "
OSError: xelatex not found on PATH, if you have not installed xelatex you may
need to do so. Find further instructions at
https://nbconvert.readthedocs.io/en/latest/install.html#installing-tex.
```

0.4 nbconvert to pure Python source code

[70]:
```
!jupyter nbconvert ch01-intro.ipynb --to python
```

```
[NbConvertApp] WARNING | Config option `kernel_spec_manager_class` not
recognized by `NbConvertApp`.
[NbConvertApp] Converting notebook ch01-intro.ipynb to python
```

```
[NbConvertApp] Writing 7281 bytes to ch01-intro.py
```

[71]:
```
!ls ch01*
```

```
ch01-intro.html  ch01-intro.ipynb  ch01-intro.pdf  ch01-intro.py
```

[ ]:
```

```