IPython

Kai Dührkop

Lehrstuhl für Bioinformatik
Friedrich-Schiller-Universität Jena
kai.duehrkop@uni-jena.de

11.-15. August 2014

- modules and libraries for scientific work in python
  - python: interactive python shell
  - numpy: vectors and matrices
  - scipy: linear algebra, optimization, statistics, much more
  - Matplotlib and pylab: plotting in python
- `http://www.scipy.org/`

ipython

- provides a REPL (read-eval-print-loop)
- but: additional syntax (so called **magics**)
- is a complete shell-replacement
- provides runtime documentation, editing and debugging features
- write your software while you execute and test it!
- qt-version is even able to embed images and plots
- autocompletion, syntax highlighting, auto-indent, and much more!

ipython

```
# starts ipython shell
ipython

# starts ipython shell in seperate
# qt window (with additional features)
ipython qtconsole

# starts ipython shell in web browser
ipython notebook

# start ipython with plotting support
ipython --pylab
```

getting help

```
# similar to pythons own help(x)
# but much nicer
list.append?
list?
5?
# use ?? for more information
os??

# without anything else,
# display ipython documentation
?
```

magics

```
# single line magics start with %
# multiline magics start with %%

%timeit sum([1 for i in range(10000)])
#=>1000 loops, best of 3: 1.12 ms per loop

%timeit sum((1 for i in xrange(10000)))
#=>1000 loops, best of 3: 414   s per loop

# like dir() but MUCH nicer
%whos
```

edit code in editor

```
# open file in editor
# enter code into python shell
# after saving it
%edit /path/to/file

# same as above, but use temp file
%edit

# edit a specific function
%edit readMassbank

# edit a specific class
%edit MyClass
```

edit code in editor

```
# you can load source code from
# a file using
%load path/to/my/file.py
```

- you can also copy/paste between editor and ipython. ipython with qtconsole will strip the **prompt** from the strings

running shell commands

```
# a line starting with ! is executed
# as shell command
!cp -r /some/files*.txt to/path

# substitute python variables into shell
# with $varname or {expression}
file = "someFile.txt"
!mv $file {file[:-4] + ".csv"}
```

running shell commands

```
# you can redirect the standard output
# to a python variable (class SList)
output = !grep "137.01023" files/*.txt

# return grep output as list
mylist = output.l
# or as string (with newlines)
mylist.n
directory = !ls files
# or as string (with spaces)
directory.s
# or as list of files
directory.f
```

Debugging

```
def myfunction(x,y):
    return x+y
# start debugger and run a certain function
%debug myfunction(1,"a")

myfunction(1, "b")
# start debugging AFTER an exception occurs!
%debug
```

Debugging

- shell debugger are always somehow clunky
- but in principal are as powerful as e.g. IntelliJ Debugger
- use **w** for printing stacktrace and **l** for printing affected source code
- **up** and **down** for moving the stacktrace up and down
- **n** for executing next line, and **s** for executing the next single operation (e.g. jumping into a function)
- create breakpoints with **b linenumber**
- **c** continues the program.
- List all other commands with **help**. Everything else is interpreted as python statements!

Other features

- running code with R using %R (also available:
- using pylab for plotting inside the shell
- ipython has a library for running python code on computer clusters

Take home messages

- using ipython instead of python shell
- develope with the ipython shell using %edit and %load
- debug with the ipython shell
- getting runtime information and help with ?
- running arbitrary shell commands with !