# Research Computing with Python, Lecture 1

Ramses van Zon

SciNet HPC Consortium

November 5, 2013

# Introduction to the Course

# About the course

- Mini graduate-style course on research computing
- Using python as the programming language.
- 4 weeks with 2 lectures per week
- Lecture from 11 am to 12 noon
- Can be taken for credit by (astro)physics grad student as modular/mini courses.
- There will be an assignment each week

**Lecture dates**

Nov 5, 7, 12, 14, 19, 21, 26, 28, 2013
11 am - 12 pm

# Course Topics

- Python programming
- Automation
- Version control
- Modular programming
- Visualization
- Selected numerical methods

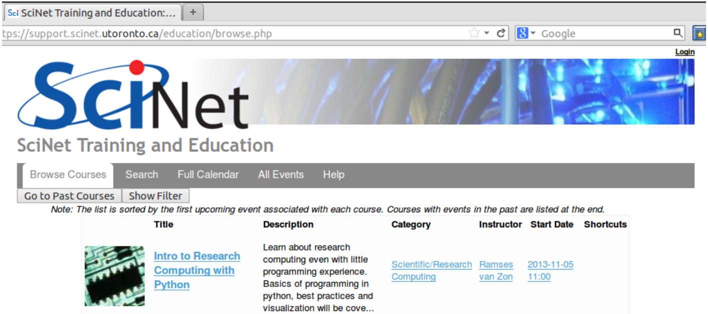
# Details

- **Prerequisites:**
  Minimal programming experience should suffice.
- **Software that you'll need:**
  Python with numpy, scipy, matplotlib and mercurial.
  Easiest to get (and preferred): Enthought Canopy
- **Instructors**
  - Ramses van Zon
  - Erik Spence
- **Grading scheme**
  The grading scheme will be based on four homework assignments, to be handed in online on the course website
- **Please fill out the sign-up sheet!**

# Course Website

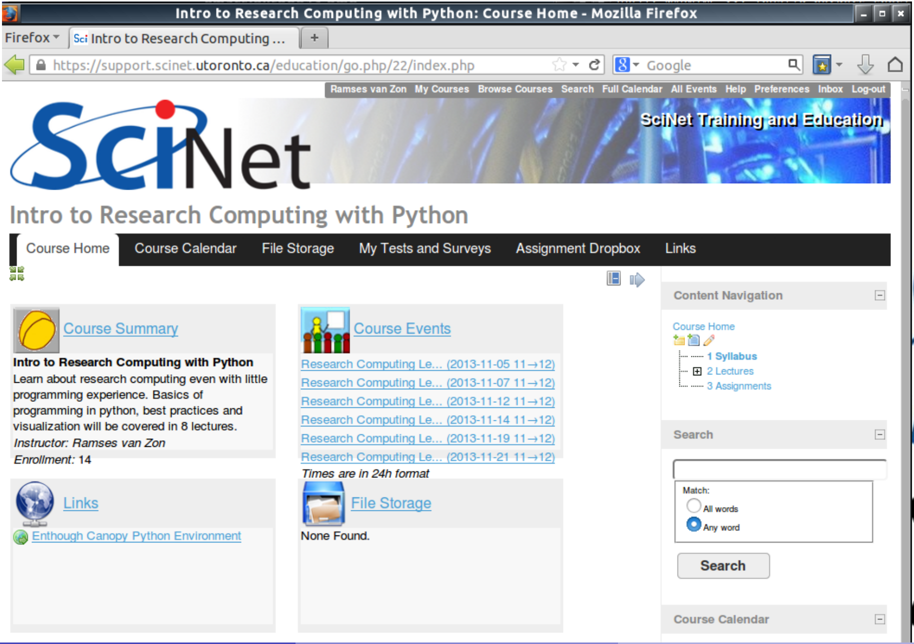
# Education Site

**https://support.scinet.utoronto.ca/education**




- Log in with your SciNet account;
  No SciNet account? Get a temporary account for the education site.
- Browse to the course site *Intro to Research Computing with Python*
  https://support.scinet.utoronto.ca/education/go.php/22/index.php
- Enroll if you have not already.

# Course website



# Course tools

On the **Course Home**, you'll see a number of *tools*:

- *Course Events:* lists upcoming lectures
- *Links:* useful web sites
- *File Storage:* pdfs of the lecture slides
- *Assignment Dropbox:* where you upload your assignments.
- *My Tests and Surveys:* your grades

*Note: On the top, there are tabs for many of the tools.*

# Course content

In the right column, you'll see the

- The content navigation, with
  - *Syllabus*
  - *Lectures*
  - *Assignment descriptions*
- Search box
- Calendar

*Note 1: you can read the content in sequence by using the gray arrows.*

*Note 2: the right column can be hidden.*

Enough preliminaries, let's get started. . .

# Research Computing with Python

# Research Computing

*A.K.A.: Computational Science, Scientific Computing.*

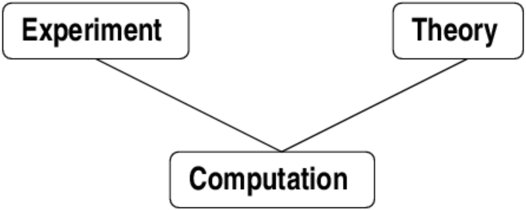
**Using a computing device (computer) to figure out numerical values of quantities of interest in the scientific endevour.**

One computes for a variety of reasons, such as

- Large data processing/data mining
- Investigating behaviour of models too complex to deal with on paper
- Interpret experimental results using a theoretical model
- Finding simpler models from more complex ones
- Visualization

# Third Leg?

*Research Computing* is often called the third leg of science:




Won't get into philosopical matters. From a practical perspective:

- Computation is used by experiment and theory.
- Research Computing can learn from best practices in both theoretical and experimental science.
- It is often closer to a well controlled experiment.
- Requires some knowledge and skills unique to computing.

# Programming

- One often needs to do a bit of programming for computing.
- Programming = telling the computer in detail what you want it to do.
- Programming languages range from low level to more abstract levels.
- Some program translates these languages into machine instructions:

**Compiler** takes your whole code and generates optimized instructions into an executable. The executable can be run afterward the compilation is done.

**Interpreter** reads a line from your code (script), generates instructions and executed them, then reads the next line, etc.

In this class, we will be doing our programming in *Python*, version 2.7 (which is what's in Canopy).

# What is Python?

- Flexible, mature (20yo) scriting-style, high-level language
- Free to use
- Ubiquitous: runs on Windows, Linux/Unix, Mac OS X
- Huge standard library, massive number of third party modules
- Much slower than C/Fortran or even IDL/MATLAB
- You should know that there is a Python 3, but because not all packages have been ported to that version, we use 2.7.

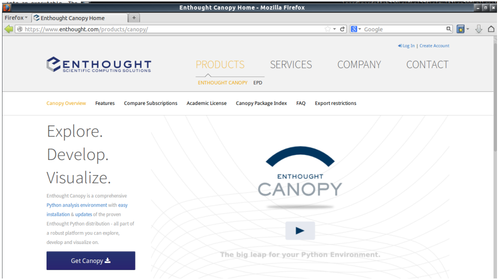
# IPython

IP[y]: IPython
Interactive Computing

- Enhanced interactive Python shell
- `--pylab`: automatically loads lots of good math, plotting stuff (Canopy loads this by default)
- If you write Python scripts, you to load these yourself
- IPython notebook: Mathematica/Maple-like IPython environment in browser.

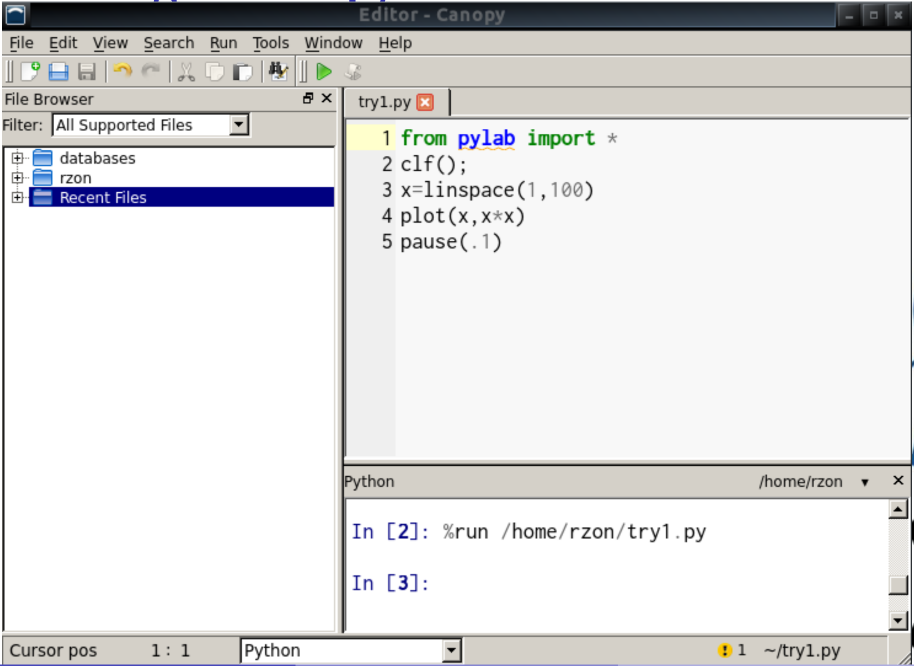
# Enthought Canopy

**https://www.enthought.com/products/canopy**

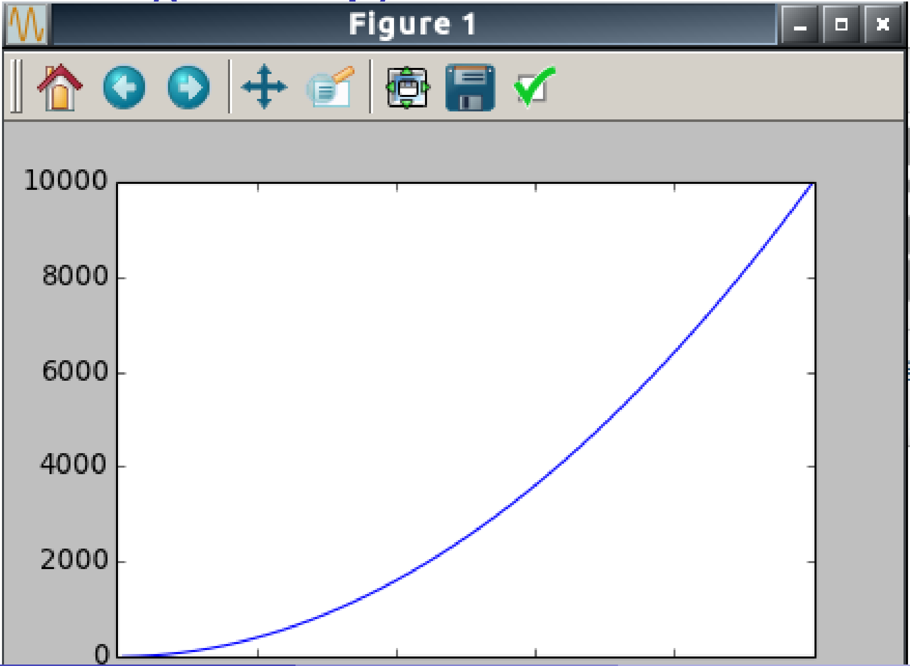



- An (I)python development and analysis environment which includes some of the more useful packages by default.
- Runs on Windows, Linux/Unix, Mac OS X
- Free version ('Express') has limited number of packages
- Academic license has more . Also free, if you register with your firstname.lastname@utoronto.ca email address.

# Enthought Canopy - Screenshot



# Enthought Canopy - Screenshot



# The Python Language

# Basic Python

- Variables
- Like most scripting languages, dont have to declare.
- Very handy for quick stuff, but has real drawbacks
- Math works the way you'd expect

```
In [1]: x=2

In [2]: y=3

In [3]: print x+y
5

In [4]: print x*y
6

In [5]: print y/x
1
```

# Variable types

Python has 5 standard data type:

- *Numbers:*
  int, long, float, complex
- *String:*
  (Single or double) quotes
- *List:*
  Square brackets
- *Tuple:*
  Parentheses. Read-only
- *Dictionary:*
  Curly braces. Unordered
  key-value list

```
In [6]: print "Hello, world!"
Hello, world!

In [7]: u=['I','am','list']

In [8]: u[1]=6

In [9] print u
['I', 6, 'list']

In [10]: v=('I','am','list' )

In [11]: w={'s':'I','p':'ls'}

In [12]: print w
{'p': 'ls', 's': 'I'}
```

# Arrays, Numpy

- Python has lists but not “real” arrays
- Arrays are supplied by numpy, automatically included by pylab
- Numpy is the backbone of most scientific computing done in Python.
- More about numpy later in course

```
In [13]: z=array([1.,2.,3.])

In [14]: print z
[ 1. 2. 3.]

In [15]: print x*z
[ 2. 4. 6.]

In [16]: z2d=array([[1.,2.],
  ...:              [4.,5.]])

In [17]: print z2d
[[ 1. 2.]
 [ 4. 5.]]
```

# Numpy, SciPy

- Numpy provides basic N-dimensional array data structure, *fast* operations on that structure.
- Some low level math libraries
- SciPy has higher-level routines - linear algebra, fftpack, sparse matrix stuff, optimization modules, etc.

SciPy.org

http://www.scipy.org/SciPy

# Python Loops

- For loops are more like *foreach*
- Each item in list
- If want a counting loop, use xrange (generates list 0..N-1)
- Note indentation: indentation is important in Python!

```
In [18]: for element in z:
   ....:     print element
   ....:
1.0
2.0
3.0

In [19]: for i in xrange(10):
   ....:     print i,
   ....:
0 1 2 3 4 5 6 7 8 9
```

# Python Functions

- Can also define functions
- 'def' keyword

```
In [20]: def squareNum(x):
   ....:     return x*x
   ....:

In [21]: print squareNum(4)
16

In [22]: print squareNum(7.3)
53.29

In [23]: print squareNum('no')
```

# If/Else

- Control flow
- Same : syntax, same punctuation significance
- Functions needn't return a value

```
In [24]: def evenOrOdd(n):
   ....:     if n % 2 == 0:
   ....:         print "even,"
   ....:     else:
   ....:         print "odd"
   ....:

In[25]: evenOrOdd(17)
odd

In[26]: evenOrOdd(18)
even,
```

# Writing Python Modules

- Can write functions in a file, import them in ipython
- specify them with filename.functionname
- Code not in functions will be run at import time.
- Use # for other comments
- Use """ in functions for documentation: docstring

```
#File: mymod.py
def myFunc(x,y):
  '''Returns sum of squares'''
  return x**2 + y**2
```

```
In [27]: import mymod.py
In [28]: help(mymod.myFunc)
Help on function myFunc in modu

myFunc(x, y)
  Returns sum of squares

In [29]: a=mymod.myFunc(1,2)
In [30]: print a
5
```

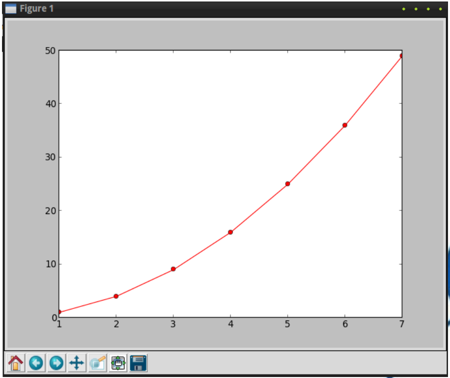
# Python Array Slicing

- Like in Fortran and MATLAB, but:
- `:` selects the entire range in that dimension
- `start:end` selects from `start` to **before** `end`
- `start:end:stride`

```
In [31]: a=[1,2,3,4,5,6]

In [32]: a[2]
Out[32]: 3

In [33]: a[:]
Out[33]: [1,2,3,4,5,6]

In [34]: a[1:3]
Out[34]: [2,3]

In [35]: a[1:6:2]
Out[35]: [2,4,6]
```

# Basic Plotting with Matplotlib

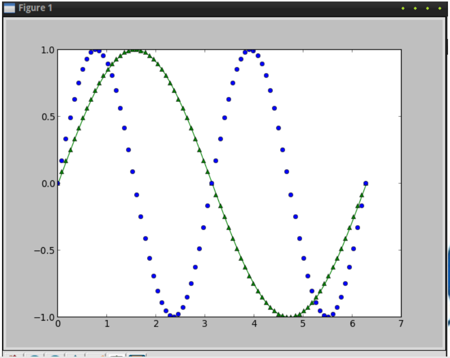
- http://matplotlib.org
- gallery of examples w/source
- MATLAB-like




```
In [36]: x=array([1.,2.,3.,4.,5.,6.,7.])

In [37]: y=x*x

In [38]: plot(x,y,'ro-')
Out[38]: [<matplotlib.lines.line2D at 0x3a8eed0>]
```

# Basic Plotting with Matplotlib

- `linspace(start,end,npnts)`
- `pi`, `e` defined
- *by default, overplot*




```
In [39]: x=linspace(0,2*pi,75)

In [40]: y=sin(x)

In [41]: z=sin(2*x)

In [42]: plot(x,y,'g^-')
Out[42]: [<matplotlib.lines.Line2D at 0x3192890>]
```

# Files

- Binary storage numpy array: `save(z)`, `load`
- Text (Ascii) storage: `loadtxt`, `savetxt`, `genfromtxt`
- Won't discuss python specific pickle format
- Other python modules can use e.g. hdf5 and other binary formats
- Can open files by hand and write out explicitly

```
In [44]: a=linspace(0,1,100)

In [45]: b=sin(a)

In [46]: save('b.npy',b)

In [47]: savetxt('b.txt',b)
```

```
In [1]: b=load('b.npy')

In [2]: c=loadtxt('b.txt'a)
```

# From IPython to Python Scripts

- Python scripts best written in pure python
- At the top, need to import modules that IPython uses:

```
from pylab import *
```

- In Canopy, scripts in the editor can be run with the ‘run’ botton.
- Graphics to screen from your script? Pure python won’t show you the graph until you do something like

```
pause(.1)
```

# Next Lecture

*Thursday November 7, 2013, 11:00 am*
**Topic: Numerics**