

Python for data science



Python for data science



Jeremy Jacobson

Visiting Assistant Professor

Institute for Quantitative Theory and Methods

jeremyallenjacobson@github.io

BRING ME



A SHRUBBERRY

We are the Knights who say ni!

COMMAND LINE

Where is the command line?

Windows

Search for cmd

Mac OS X

The Mac command line is a program called Terminal. It lives in the folder

Linux

You already know the answer

/Applications/Utilities/

Go to the command line

```
Microsoft Windows [Version 10.0.10586]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\jajaco3>
```

The letter may be different...so you might see something like:

```
D:\YourName\Projects\Python>
```

From now on we will denote the command line by a dollar sign.

```
$
```

Command syntax

Windows	Mac OS X	Linux
dir	ls	ls
cd	cd	cd
cd..	cd ..	cd ..
cd\	cd\	cd\
mkdir	mkdir	mkdir
rmdir	rm	rm

```
$ conda
usage: conda-script.py [-h] [-V] [--debug] command ...
conda is a tool for managing and deploying applications, environments and packages.
Options:
positional arguments:
  command
    info          Display information about current conda install.
```

```
$ conda info
Current conda install:

      platform : win-64
      conda version : 4.1.6
      conda-env version : 2.5.1
      conda-build version : 1.21.3
      python version : 2.7.12.final.0
      requests version : 2.10.0
      root environment : C:\Users\jajaco3\AppData\Local\Continuum\Anaconda2 (writable)
      default environment : C:\Users\jajaco3\AppData\Local\Continuum\Anaconda2
      envs directories : C:\Users\jajaco3\AppData\Local\Continuum\Anaconda2\envs
      package cache : C:\Users\jajaco3\AppData\Local\Continuum\Anaconda2\pkgs
      channel URLs : https://repo.continuum.io/pkgs/free/win-64/
                      https://repo.continuum.io/pkgs/free/noarch/
                      https://repo.continuum.io/pkgs/pro/win-64/
                      https://repo.continuum.io/pkgs/pro/noarch/
      config file : None
      offline mode : False
      is foreign system : False
```

What if I need to reproduce someone elses calculation using specific versions of various packages?

```
$ conda create --name reproduceEnv python=2.7 bokeh=0.12.3

Fetching package metadata .....
Solving package specifications: .....

Package plan for installation in environment C:\Users\jajaco3\AppData\Local\Continuum\Anaconda

The following NEW packages will be INSTALLED:

backports:           1.0-py27_0
backports_abc:        0.4-py27_0
bokeh:               0.12.3-py27_1
futures:             3.0.5-py27_0
jinja2:              2.8-py27_1
markupsafe:          0.23-py27_2
mkl:                 11.3.3-1
numpy:               1.11.2-py27_0
pip:                 8.1.2-py27_0
python:               2.7.12-0
python-dateutil:     2.5.3-py27_0
pyyaml:              3.12-py27_0
requests:             2.11.1-py27_0
setuptools:           27.2.0-py27_1
singledispatch:       3.4.0.3-py27_0
six:                 1.10.0-py27_0
ssl_match_hostname:  3.4.0.2-py27_1
tornado:              4.4.2-py27_0
vs2008_runtime:       9.00.30729.1-2
wheel:                0.29.0-py27_0

Proceed ([y]/n)?
```

If you were to hit 'y' you would see this:

```
Fetching packages ...
numpy-1.11.2-p 100% | #####| Time: 0:00:00 7.95 MB/s
pyyaml-3.12-py 100% | #####| Time: 0:00:00 1.10 MB/s
requests-2.11. 100% | #####| Time: 0:00:00 3.06 MB/s
setuptools-27. 100% | #####| Time: 0:00:00 3.83 MB/s
tornado-4.4.2- 100% | #####| Time: 0:00:00 2.69 MB/s
bokeh-0.12.3-p 100% | #####| Time: 0:00:00 5.53 MB/s
Extracting packages ...
[      COMPLETE      ]|#####| 100%
Linking packages ...
[      COMPLETE      ]|#####| 100%
#
# To activate this environment, use:
# > activate reproduceEnv
#
# To deactivate this environment, use:
# > deactivate
#
$
```

To verify that the 'reproduceEnv' environment has now been added,
type the command:

```
$ conda info --envs

# conda environments:
#
reproduceEnv          C:\Users\jajaco3\AppData\Local\Continuum\Anaconda2\envs\reproduceEnv
root                  * C:\Users\jajaco3\AppData\Local\Continuum\Anaconda2
```

Activate this environment with:

MAC/LINUX:

```
$ source activate reproduceEnv
```

WINDOWS:

```
C:\>activate reproduceEnv
(reproduceEnv) C:\>
```

Verify the correct versions with:

```
(reproduceEnv) C:\>python --version
Python 2.7.12 :: Continuum Analytics, Inc.

(reproduceEnv) C:\>bokeh --version
0.12.3

(reproduceEnv) C:\>
```

To return to Anaconda installed defaults, deactivate this environment with the command:

MAC/LINUX: \$ source deactivate reproduceEnv

WINDOWS: (reproduceEnv) C:\>deactivate reproduceEnv
C:\>

Verify the Anaconda installed default version with:

```
C:\>python --version
Python 2.7.12 :: Anaconda 4.1.1 (64-bit)

C:\>bokeh --version
0.12.0

C:\>
```

To list packages available use:

```
$ conda list
# packages in environment at C:\Users\jajaco3\AppData\Local\Continuum\Anaconda2:
#
_nb_ext_conf      0.2.0          py27_0
alabaster         0.7.8          py27_0
anaconda          4.1.1          np111py27_0
anaconda-client    1.4.0          py27_0
anaconda-navigator 1.2.1          py27_0
argcomplete        1.0.0          py27_1
astropy            1.2.1          np111py27_0
babel              2.3.3          py27_0
backports          1.0            py27_0
backports_abc      0.4            py27_0
beautifulsoup4     4.4.1          py27_0
bitarray           0.8.1          py27_1
blaze               0.10.1         py27_0
bokeh              0.12.0         py27_0
boto                2.40.0         py27_0
bottleneck         1.1.0          np111py27_0
bzip2               1.0.6          vc9_3  [ vc9 ]
cdecimal           2.3            py27_2
cffi                1.6.0          py27_0
chest               0.2.3          py27_0
click               6.6            py27_0
cloudpickle        0.2.1          py27_0
clyent              1.2.2          py27_0
colorama           0.3.7          py27_0
comtypes            1.1.2          py27_0
conda              4.1.6          py27_0
```

What is Python?

- Type commands individually into **interpreter**
- Create a **script** file (myPythonFile.py).



If you want to type commands individually,
start the **Python interactive shell** by
typing `python`.

```
$ python

Python 2.7.12 |Anaconda 4.1.1 (64-bit)| (default, Jun 29 2016, 11:07:13) [MSC v.1500 64 bit (I
Type "help", "copyright", "credits" or "license" for more information.
Anaconda is brought to you by Continuum Analytics.
Please check out: http://continuum.io/thanks and https://anaconda.org

>>>
```

Python Interactive Shell



```
$ python

Python 2.7.12 |Anaconda 4.1.1 (64-bit)| (default, Jun 29 2016, 11:07:13) [MSC v.1500 64 bit (A
Type "help", "copyright", "credits" or "license" for more information.
Anaconda is brought to you by Continuum Analytics.
Please check out: http://continuum.io/thanks and https://anaconda.org

>>>
```

```
>>> 2+2  
4  
>>> 53/3  
17  
>>> 53/4  
13  
  
>>> 53/4.0  
13.25  
>>> 53.0/4  
13.25
```

The integer numbers (e.g. 2, 4, 20)
have type: **int**

The decimals (e.g. 53.0, 4.0, 13.25)
have type: **float**

If both operands are of type int, floor
division is performed and an int is
returned (e.g. 53/4 returns 13)

If either operand is a float, classic
division is performed and a float is
returned (e.g. 53.0/4 or 53/4.0 return
13.25)

```
>>> type(53/3)
<type 'int'>
>>> type(53)
<type 'int'>
>>> type(53.0)
<type 'float'>
```

comment

```
>>> 53 / 4 # int / int -> int
13
>>> 53 / 4.0 # int / float -> float
13.25
>>> 53 // 4
13
>>> 53 // 4.0
13.0
>>> 53 // 4.0 # explicit floor division discards the remainder
13.0
>>> 53 % 4 # the % operator returns the remainder
1
>>> 13 * 4 + 1 # result * divisor + remainder
53
```

With Python, use the `**` operator to calculate powers

```
>>> 9 ** 1  
9  
>>> 9 ** 2  
81  
>>> 9 ** 3  
729  
>>> 9 ** 4  
6561  
>>> 9 ** 5  
59049
```

You can also use `pow(,)`

```
>>> pow(2,3)  
8  
>>> pow(9,1)  
9  
>>> pow(9,2)  
81  
>>> pow(9,3)  
729  
>>>
```

The equal sign (=) is used to assign a value to a variable.
Afterwards, no result is displayed before the next cell.

```
>>> width = 23.56  
>>> height = 46.9  
>>>
```

If a variable is not “defined” (assigned a value), trying to use it will give you an error:

```
>>> heitgh * width  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
NameError: name 'heitgh' is not defined  
>>> height * width  
1104.964
```

In interactive mode, the last printed expression is assigned to the variable `_`. This means that when you are using Python as a desk calculator, it is somewhat easier to continue calculations, for example:

```
>>> width = 23.5482934
>>> height = 46.923
>>> height * width
1104.9565712081999
>>>
>>> _
1104.9565712081999
>>> round(_,4)
1104.9566
>>> round(_,2)
1104.96
>>>
```

$$\binom{n}{k} = \frac{n!}{(n-k)!k!}$$

```
>>> factorial(10)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'factorial' is not defined

>>> math.factorial(10)
3628800
>>> math.factorial(2)*math.factorial(8)
80640
>>> math.factorial(10)/_
45
```

$$\binom{10}{2} = \frac{10!}{(10-2)!2!} = 45$$

$$\cos(\pi) = -1$$

```
>>> cos(pi)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'cos' is not defined
>>> math.cos(math.pi)
-1.0
```

$$10! = 3628800$$

$$\cos(\pi) = -1$$

```
>>> from math import cos, pi, factorial  
  
>>> factorial(10)  
3628800  
>>> cos(pi)  
-1.0
```

$$10! = 3628800$$

$$\cos(\pi) = -1$$

$$e^1 = 2.71828182845904509079559$$

```
>>> from math import *
>>> factorial(10)
3628800
>>> cos(pi)
-1.0
>>> exp(1)
2.718281828459045
```

$$e^1 = 2.71828182845904509079559829842764884233474731445312$$

```
>>> exp(1)
2.71828182845904509079559829842764884233474731445312

>>> format(exp(1), '.50g')
'2.7182818284590450907955982984276488423347473144531'

>>> format(exp(1), '.51g')
'2.71828182845904509079559829842764884233474731445312'

>>> format(exp(1), '.52g')
'2.718281828459045090795598298427648842334747314453125'

>>> format(exp(1), '.53g')
'2.718281828459045090795598298427648842334747314453125'
```

Python as a desktop calculator: fractions module

$$\frac{a}{b} + \frac{c}{d} = \frac{ad+bc}{bd}$$

$$\frac{a}{b} \times \frac{c}{d} = \frac{ac}{bd}$$

```
>>> from fractions import Fraction  
  
>>> Fraction(1,6)  
Fraction(1, 6)  
>>>  
Fraction(1, 6)  
  
>>> float(Fraction(1,6))  
0.1666666666666666  
  
>>> Fraction(1,6)+Fraction(1,6)  
Fraction(1, 3)
```

$$\frac{1}{6} \times \frac{1}{6} = \frac{1}{36}$$

$$\left(\frac{1}{6}\right)^2 = \frac{1}{36}$$

```
>>> Fraction(1,6)*Fraction(1,6)
Fraction(1, 36)
>>> Fraction(1,6)**2
Fraction(1, 36)
>>> pow(Fraction(1,6), 2)
0.02777777777777776
```

$$\left(\frac{1}{6}\right)^{15} = \frac{1}{470184984576}$$

```
>>> Fraction(1,6)**15
Fraction(1, 470184984576)
>>> float(_)
2.1268224907304786e-12
```

$$\frac{35}{19238} \times \frac{12}{12384} = \frac{35}{19853616}$$

```
>>> Fraction(35,19238)*Fraction(12,12384)
Fraction(35, 19853616)
>>>>> float(_)
1.7629030399298546e-06
>>>
```



```
# Python 3: Fibonacci series up to n
>>> def fib(n):
>>>     a, b = 0, 1
>>>     while a < n:
>>>         print(a, end=' ')
>>>         a, b = b, a+b
>>>     print()
>>> fib(1000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 98
```



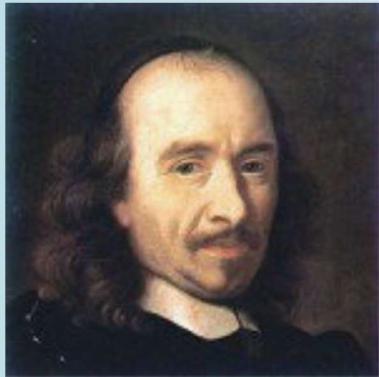
```
# Python 3: Fibonacci series up to n
>>> def fib(n):
>>>     a, b = 0, 1
>>>     while a < n:
>>>         print(a, end=' ')
>>>         a, b = b, a+b
>>>     print()
>>> fib(1000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 98
```

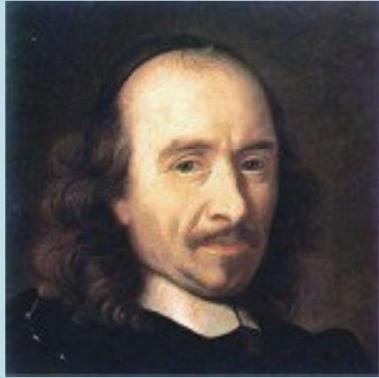
3 types of Functions

- functions always available for usage
- functions contained within external modules which must be imported
- functions defined by a programmer with the `def` keyword (see example on left).



Defining functions: Le Chevalier de Méré





$$P(\text{throw a pair of sixes}) = \frac{1}{36}$$

$$P(\text{do not throw a pair of sixes}) = \frac{35}{36}$$

$$P(\text{at least one pair of sixes in 24 throws}) = ?$$



$$\begin{aligned} &= 1 - P(\text{no pair of sixes in 24 throws}) \\ &= 1 - \left(\frac{35}{36}\right)^{24} \end{aligned}$$



Defining functions

$$P(\text{at least one pair of sixes in } n \text{ throws}) = \\ = 1 - \left(\frac{35}{36}\right)^n$$

```
>>> def chevalier(n):
...     return Fraction(1,1)-Fraction(35,36)**n
...
>>>
```



Defining functions

```
>>> chevalier(24)
Fraction(11033126465283976852912127963392284191, 22452257707354557240087211123792674816)
>>>
```

$$\begin{aligned} P(\text{at least one pair of sixes in 24 throws}) &= \\ &= \frac{11033126465283976852912127963392284191}{22452257707354557240087211123792674816} \end{aligned}$$



Defining functions

```
>>> chevalier(24)
Fraction(11033126465283976852912127963392284191, 22452257707354557240087211123792674816)
>>> float(_)
0.49140387613090325
>>>
```

$$\begin{aligned} P(\text{at least one pair of sixes in 24 throws}) &= \\ &= \frac{11033126465283976852912127963392284191}{22452257707354557240087211123792674816} \\ &= 0.49140387613090325 \end{aligned}$$



Defining functions

```
>>> chevalier(25)
Fraction(408611683992293747092011689842522621501, 808281277464764060643139600456536293376)
>>> float(_)
0.5055315462383781
>>>
```

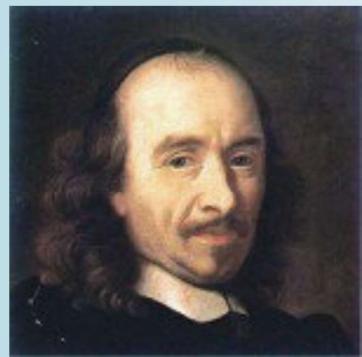
$$\begin{aligned}P(\text{at least one pair of sixes in 25 throws}) &= \\&= \frac{408611683992293747092011689842522621501}{808281277464764060643139600456536293376} \\&= 0.5055315462383781\end{aligned}$$



Defining functions: exit()

```
>>> exit()

C:\Users\jajaco3>python
Python 2.7.12 |Anaconda 4.1.1 (64-bit)| (default, Jun 29 2016, 11:07:13) [MSC v.1500 64 bit (A
Type "help", "copyright", "credits" or "license" for more information.
Anaconda is brought to you by Continuum Analytics.
Please check out: http://continuum.io/thanks and https://anaconda.org
```



"Non défini!"

```
>>> chevalier(25)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'chevalier' is not defined
>>>
```



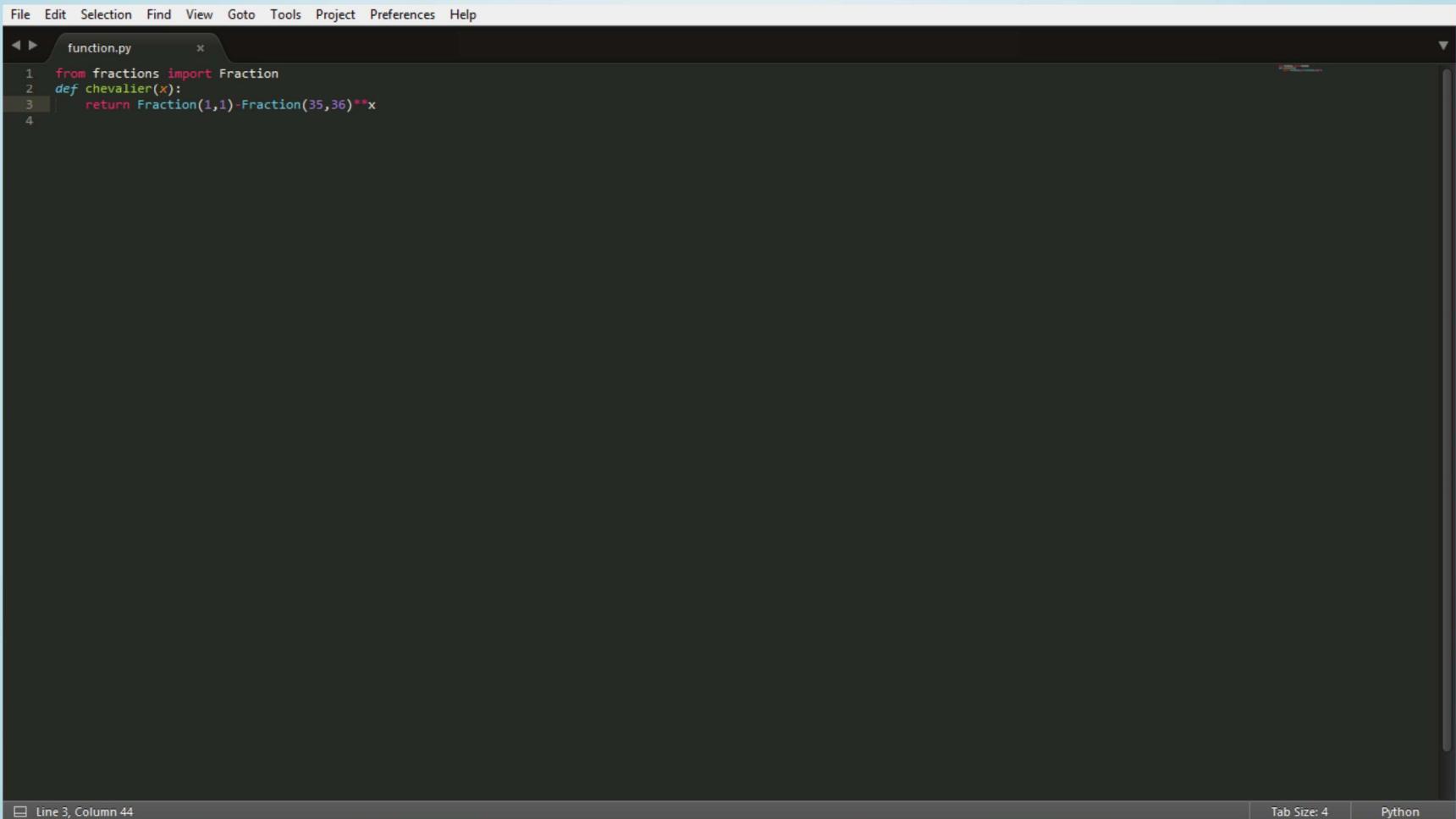
Writing Python code

If you want to reuse your function, you must write it down:

- In a .py file using a text editor, e.g. sublime text
- In a Ipython/Jupyter notebook



Writing Python code: Sublime text



A screenshot of the Sublime Text code editor. The menu bar at the top includes File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, and Help. A tab labeled "function.py" is open. The code in the editor is:

```
1 from fractions import Fraction
2 def chevalier(x):
3     return Fraction(1,1)-Fraction(35,36)**x
4
```

The status bar at the bottom shows "Line 3, Column 44", "Tab Size: 4", and "Python".



Writing Python code: Jupyter Notebooks

Jupyter Lecture2 Last Checkpoint: 09/10/2016 (autosaved)

Notebook saved | Python [default] O

File Edit View Insert Cell Kernel Help

Slide Type Slide ▾

Python for data science lecture 2

Slide Type ▾

Jeremy Jacobson (jeremy.a.jacobson@emory.edu) <https://github.com/jeremyallenjacobson>

The latest version of this [Jupyter notebook](#) lecture is available at <https://github.com/jeremyallenjacobson/PythonForDataScience>.

Slide Type ▾

Python program files

Slide Type Sub-Slide ▾

* Python code is usually stored in text files with the file ending ".py":
`chevalier.py`

Slide Type ▾

Jupyter notebooks

Slide Type ▾



Modules

- A module is a file containing Python definitions and statements.
- The file name is the module name with the suffix .py appended.
- For instance, use your favorite text editor to create a file called mymodule.py in the current directory with the following contents:

```
# Probability problem of Chevalier module
from fractions import Fraction
def chevalier(x):
    return Fraction(1,1)-Fraction(35,36)**x
def fchevalier(x):
    return float(Fraction(1,1)-Fraction(35,36)**x)
```



Modules

Now enter the Python interpreter and import this module with the following command:

```
C:\Users\jajaco3\Desktop\PythonForDataScience>python
Python 2.7.12 |Anaconda 4.1.1 (64-bit)| (default, Jun 29 2016, 11:07:13) [MSC v.1500 64 bit (A
Type "help", "copyright", "credits" or "license" for more information.
Anaconda is brought to you by Continuum Analytics.
Please check out: http://continuum.io/thanks and https://anaconda.org
>>> import mymodule
```



Modules

Using the module name you can access the functions:

```
>>> mymodule.chevalier(24)
Fraction(11033126465283976852912127963392284191, 22452257707354557240087211123792674816)
>>> mymodule.fchevalier(24)
0.49140387613090325
>>>
```



Modules

If you intend to use a function often you can assign it to a local name:

```
>>> chev = mymodule.chevalier
>>> chev(25)
Fraction(408611683992293747092011689842522621501, 808281277464764060643139600456536293376)
>>> fchev = mymodule.fchevalier
>>> fchev(25)
0.5055315462383781
>>>
```



Modules

The built-in function `dir()` is used to find out which names a module defines.

It returns a sorted list of strings:

```
>>> import mymodule
>>> dir(mymodule)
['Fraction', '__builtins__', '__doc__', '__file__', '__name__', '__package__', 'chevalier', 'f
>>>
```



Modules

- The built-in function `help()` provides helpful information from "docstrings" written into the function.

```
>>> import math
>>> help(math.log)
Help on built-in function log in module math:

log(...)
    log(x[, base])

    Return the logarithm of x to the given base.
    If the base not specified, returns the natural logarithm (base e) of x.

>>>
```

- However, not all functions have docstrings (e.g. we did not write one for our function) but the vast majority are documented this way.



Modules: packages

- Packages are a way of structuring Python's module namespace by using "dotted module names".
- For example, the module name A.B designates a submodule named B in a package named A.
- The use of dotted module names saves the authors of multi-module packages like NumPy from having to worry about each other's module names.