functions (part 2) and modules

Ben Bolker

22:29 26 January 2017

Functions calling functions

- You can pass anything to a function as an argument (even a function!)

```
def repeat_fun(f,startval,n):
    """Given a function f and a starting value startval,
    apply the function n times (each time using the previous
    result as input
    """
    y = startval
    for i in range(n):
        y=f(y)
    return(y)

def sqr(x):
    return(x*x)

repeat_fun(sqr,3,3)
```

Scope

- Where does Python look for things?
- What happens here?

```
z = 1
def add_z(x):
    return(x+z)

add_z(z)
```

- **LEGB** (Local, Enclosing, Global, Built-in)
  - *Local*: symbols defined in the function, and arguments
  - *Enclosing*: symbols defined *in the function within which this function was defined*
  - *Global*: elsewhere in the file/module
  - *Built-in*: Python keywords

Modules

Collections of functions you might want to use.

importing

- `import`
- refer to functions via module prefix
- `import VeryLongModuleName as vlmn`: use abbreviation

- can import just one or two functions: `from math import sqrt, log`
- can import everything (but usually don't): `from <module> import *`
- can import *your own modules* (i.e., functions in a `.py` file)

finding out about modules

- `help("modulename")`
- official modules
- list of useful modules
- some modules we will definitely be using:
  - `math`: basic math functions
  - `matplotlib`: drawing pictures
  - `random`: picking random numbers
  - `numpy`: numerical computation in general (e.g. linear alg and calculus)
  - `pandas`: data analysis
- slightly less useful but maybe using:
  - `nose`: code testing framework
  - `scipy`: even more scientific computing tools
  - `cmath`: math functions handling complex numbers
  - `re`: regular expressions
  - `sympy`: symbolic computation
  - `timeit`: how long does my code take?