# *functions and modules*

*Ben Bolker*

*22:57 22 January 2015*

## *Functions*

Reference: Python tutorial section 4.6

- *the* most important tool for structuring programs
- allows *modularity*
- basic definition: `def function_name(args):` plus indented code block
- when function is called, go to the function, with the arguments, run code until you hit `return()` (return `None` if you get to the end without a `return`)

### Return values

- most functions return values
- might not ... *side effects*
  - changing a (mutable!) variable
  - input/output (create a plot, write output to a file, turn on a machine, ...)

### Function arguments

- basic arguments: *unnamed*, *mandatory*
- think of them as dummy variables; could be the same or different from the name in the calling environment

```
def foo(x):
    x += 1
    return(x)
x = 2
print(foo(x))
print(x)
z = 2
print(foo(z))
print(z)
```

```
## 3
## 2
## 3
## 2
```

Since `z` is a number (**immutable**), it doesn't change; if you want it to change, use `z=foo(z)`

Changes within functions follow **the same rules we learned on Wednesday**.

Compare:

```
def bar(x):
    x = [2,3,4]
    return(None)

z = [1,2,3]
bar(z)
z
```

With:

```
def bar(x):
    x[0] = 7
    return(None)

z = [1,2,3]
bar(z)
z
```

Functions can call functions (even themselves, **recursively**):

```
def factorial(x):
    if (x==1):
        return(1)
    return(x*factorial(x-1))

factorial(5)
```

What happens if we forget to put in the `if` clause?

- More often, functions call functions
- You can pass anything to a function as an argument (even a function!)

```
def repeat_fun(f,x,n):
    for i in range(n):
        x=f(x)
```

```
    return(x)

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

repeat_fun(sqr,3,3)
```

Scope

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

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

foo(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

Function arguments, continued

- **optional** arguments: give *default* values
  - e.g. logarithm: `def log(value,math.e)`
- **named** arguments:
- matching by position vs. name

*Docstrings*

- use them!

## *Modules*

importing

- `import`

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

finding out about

- official modules
- list of useful module
- `math`, `cmath`, `re`, `random`, `numpy`, `scipy`, `matplotlib`, `timeit`