functions and modules

Ben Bolker

17:51 22 January 2017

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
- inputs are called **arguments**. outputs are called **return values**
- 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*
  - input/output (create a plot, write output to a file, turn on a machine, …)
  - changing a (mutable!) variable

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

Changes within functions follow the standard mutability rules:

Compare:

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

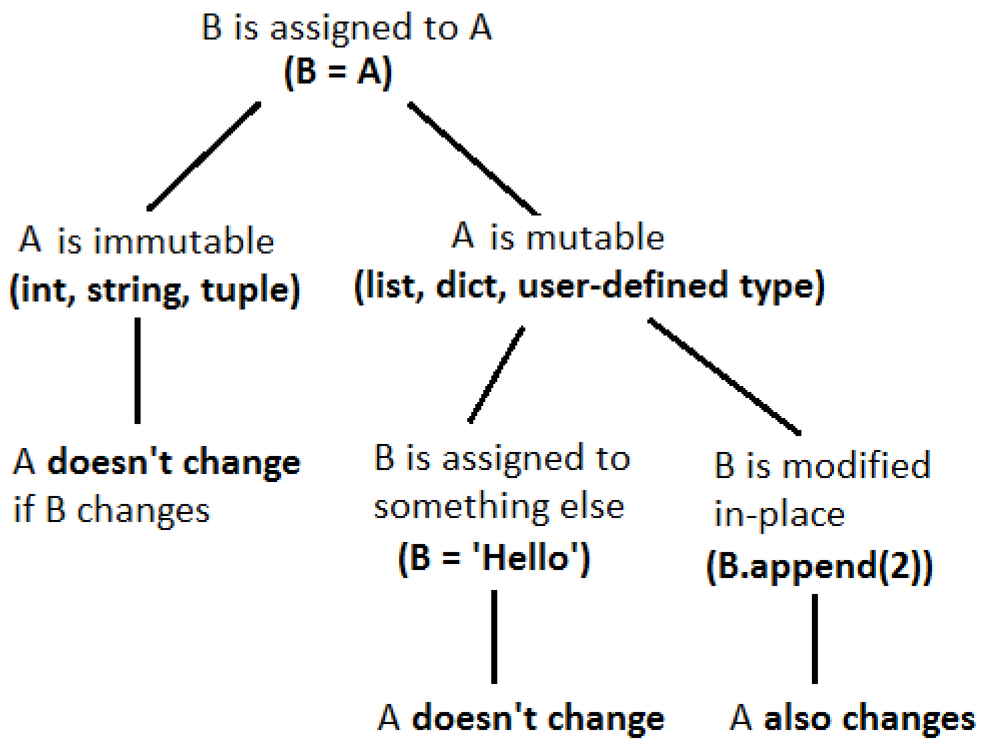

Figure 1: mutability mnemonic

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

With:

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

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

- **optional** arguments: give *default* values
    - e.g. logarithm: `def log(value,math.e)`

Docstrings

- always say something about what your function does. (Feel free to give me a hard time in class if I don't.)

```
def documented_function():
    """this is a function that does
       nothing very useful
    """
    return(None)
```