Python: Functions & Modules

Kai Dührkop

Lehrstuhl für Bioinformatik
Friedrich-Schiller-Universitaet Jena
kai.duehrkop@uni-jena.de

11.-15. August 2014

Section 1

Functions

- keyword **def** introduces a function definition
- first statement may be a **docstring**

Defining and calling a function

```
def gcd(a,b):
    '''Euclidean algorithm'''
    while b != 0:
      h = a % b
      a, b = b, h
    return a

gcd(42,23)
```

- docstrings are used for generating documentation (like javadoc)
- help(function) or help(klass.method) displays docstring

Function Documentation

```
help(set.add)
#==> add(...)
#     Add an element to a set.
#     This has no effect if the element
#     is already present.
```

- dir displays all properties and methods of the argument
- without arguments: displays all variables and functions in local scope
- type returns the type of the argument
- better: property __class__, works also for objects

help

```
x = 5
y = [1,2,3]
dir() #=> x, y, ...
dir(y) #=> append, pop, ...
type(y) #=> list
y.__class__ #=> list
help(list.append)
```

- function names are symbols
- symbols can be reassigned
- but most builtin functions are keywords instead of functions (e.g. print)

Renaming a function

```
def gcd(a, b):
    ...

g = gcd
g42 = gcd(42, 23)

g(42, 23) #this works!
g42       #same result!
```

- arguments may have default values

Default argument values

```
def gcd(a=42, b=23):
    ...

gcd()
gcd(42, 23)
gcd(a=42, b=23) #same results!
```

Defining and calling functions with variable arguments

```
def gcd(a, b, *args, **d):
    ...

gcd(42, 23,
    'The', 23, 'enigma'                    # *args
    question='everything', answer=42 # **d
    )
```

- *args is a **tuple**
- **d is a **dict**

- functions often return **tuple**

Function returning tuple

```
def divide_with_rest(a,b):
    return (a/b, a%b)

t, r = divide_with_rest(5,2)
```

Variable scope

```
a = 3

def func():
    global b
    a, b, c = 1, 2, 3

func()
print a, b                  #result: 3, 2
print c                     #error
```

- variables defined in functions are local
- **global** keyword to define globals in functions **(BAD STYLE!)**

- functions can be defined inside of functions

Local functions

```
def fibonacci(n):
    if n == 1: return 1
    if n == 2: return 2
    def helper(a, b, k):
        if k <= 0: return a+b
        else: return helper(b, a+b, k-1)
    return helper(1, 2, n-3)
```

Section 2

Modules

Package Structure

```
package/                          #top level package
    __init__.py
    subpackage/                   #subpackage
        __init__.py
        module.py
        module2.py
```

- a module is a python file
- the interpreter interprets directories on the **sys.path** with an **__init__.py** as package of modules
- **__init__.py** is loaded first, may contain initialization code

Importing and using modules

```
import sys                      #import sys module
from os import chdir            #import chdir function
                                #from os module

chdir(sys.argv[1])
```

- module import: call functions/variables by module & name
- function/variable import: call functions/variables by name

CAUTION!

```
from package import *
```

- **does not import all functions of a module!**
- imports functions defined in:

__init__.py

```
__all__ = ['function', 'function2']
```

Useful stuff

```
import sys, __builtin__

print sys.__name__          #print module name

print sys.__version__       #print module version

print dir(__builtin__)      #list all built-in
                            #functions and
                            #variables
```

- **there is no standard main function!**
- python executes all lines outside of functions!
  - what do we do if we want to import a module without executing the "main code"?

Use standard boilerplate code:

```
if __name__=="__main__":
    ...                          #"main code" here
```

Or:

```
def main:
    ...                          #"main code" here

if __name__=="__main__":
    main()
```

- Program control
  - **sys.argv** cmd line args (argv[0] is the script name)
  - **sys.exit([arg])** exit with exit code [arg] (int, string or object)
- Numbers
  - **sys.float_info** information about the float type
  - **sys.maxint** max integer
  - **sys.maxsize** max size of containers
- Environment
  - **sys.modules** currently loaded modules
  - **sys.path** module search path
  - **sys.platform** operating system info

Take home messages

- Functions
  - dir, type, __class__ and help for getting informations at runtime
  - function arguments may have default values
  - functions often return **tuple**
- Modules
  - **import** module or **from** module **import** function
  - best practice: **main** boilerplate code
  - **sys** module for program control, number and environment information