Python for Scientific Data Analysis

Basic Python

Section 3: Functions

Usually, when people talk about functions in Python, they mean the thing that starts with a `def` command.

Functions do three things:

1. They name pieces of code the way variables name strings and numbers.
2. They take arguments the way your scripts take argv.
3. Using 1 and 2 they let you make your own ”mini-scripts” or ”tiny commands.”

Variables and Functions

Here is an example, ex18.py:

```
#definitions ...

def print_two(*args):

        arg1,arg2 = args
        print("arg1: %r, arg2: %r" % (arg1,arg2))

def print_two_again(arg1,arg2):
        print("arg1: %r, arg2: %r" % (arg1,arg2))

def print_one(arg1):
        print("arg1: %r" % arg1)

def print_none():
        print("nuthin'.")


print_two("James","Kirk")
print_two_again("James","Kirk")
print_one("Khan!!!")
print_none()
```

If you execute this module, you will first see `print_two` run with variables `James` and `Kirk` and then `print_two_again` also with variables `James` and `Kirk`. Finally, `print_none` is run without arguments. You should then see:

```
arg1: 'James', arg2: 'Kirk'
arg1: 'James', arg2: 'Kirk'
arg1: 'Khan!!!'
nuthin'.
```

In this module, multiple python functions are defined and then are run with different variables. But what if you are only interested in output generated from one function? Well, you could manually edit the last few lines, commenting out all but one function call. Or, you comment out the last three lines entirely but instead 1) start the python prompt (by typing `python` in the terminal window), 2) *import* the function, and 3) call it with a variable name.

CAUTION If you run a piece of python code from terminal (e.g. `python [name of file]`) then that piece of code does not have to be in your current working directory. **BUT** If you import a function from a file, that file has to be in your current working directory or in your `$PYTHONPATH`.

Where is your path? This will help:

```
print('\n'.join(sys.path))
```

For the below examples, for importing functions, we will assume we are in the current working directory. An example with an edited module, ex18b.py:

```
#definitions, modified ...

def print_two(*args):

        arg1,arg2 = args
        print("arg1: %r, arg2: %r" % (arg1,arg2))

def print_two_again(arg1,arg2):
        print("arg1: %r, arg2: %r" % (arg1,arg2))

def print_one(arg1):
        print("arg1: %r" % arg1)

def print_none():
        print("nuthin'.")
```

And then run as ...

```
>>> from ex18b import print_one
>>> print_one("Picard!")
arg1: 'Picard!'
```

You can also import multiple Python functions:

```
>>> from ex18b import print_one,print_two
>>> print_one("Picard!")
arg1: 'Picard!'
>>> print_two("Jean-Luc","Picard")
arg1: 'Jean-Luc', arg2: 'Picard'
```

Now, you can have as many Python functions within a module as you want. Want to know the names of all the functions in a module? Try this one neat trick:

```
>>> import inspect
>>> import [module name]
>>> functions=inspect.getmembers([module name],inspect.isfunction)
>>> for a in range(len(functions)): print(functions[a])
```

Used in the example for ex18b, we get ...

```
('print_none', <function print_none at 0x1099f8830>)
('print_one', <function print_one at 0x1099f8200>)
('print_two', <function print_two at 0x109978050>)
('print_two_again', <function print_two_again at 0x1099f19e0>)
```

There are other, simpler ways of printing out function lists, albeit in an unformatted way: `dir([module name])`, which for ex18b returns ...

```
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__na
me__', '__package__', '__spec__', 'print_none', 'print_one', 'print_two',
 'print_two_again']
```

Files and Functions

Now see what happens when we manipulate files within functions, as in ex20.py

```
from sys import argv

script,input_file = argv

def print_all(f):

        print(f.read())
def rewind(f):
        f.seek(0)

def print_a_line(line_count,f):
        print( "hihi", line_count,f.readline())

current_file=open(input_file)

print("First let's print the whole file:\n")
print_all(current_file))

print("now rewind")
rewind(current_file)

print("print 3 lines")

current_line=1

print_a_line(current_line,current_file)

current_line=current_line+1

print_a_line(current_line,current_file)

current_line=current_line+1
print_a_line(current_line,current_file)
```

If we call ex20.py using our old friend -- `python ex20.py ex15_sample.txt` -- we get the following:

```
First let's print the whole file:

This is stuff I typed into a file.
It is really cool stuff.
Lots and lots of fun to have in here.

now rewind
print 3 lines
hihi 1 This is stuff I typed into a file.

hihi 2 It is really cool stuff.

hihi 3 Lots and lots of fun to have in here.
```

In this example, we read out the file in full: i.e. pointing to the end. The *seek* function re-sets the file's current position at a given offset (an argument of the function *seek*). We set it back to the beginning in this example. Then we have three separate calls to the function *print_a_line* . *print_a_line* just reads one line and prints to output. The argument *current_line* doesn't actually do anything in this function call: it's just for housekeeping.

Sending and Returning Values from Functions

Now, we will do an example where we send and return values from functions after importing a module. The module ex25.py looks like ...

```
#import this

def break_words(stuff):
        """This function will break up words."""
        words = stuff.split(' ')
        return words
def sort_words(words):
        """Sorts."""
        return sorted(words)

def print_first_word(words):
        """Prints the first word"""
        word=words.pop(0)
        print(word)

def print_last_word(words):
        """Prints the last word"""
        word=words.pop(-1)
        print(word)
def sort_sentence(sentence):
        """Takes in a full sentence"""
        words = break_words(sentence)
        return sort_words(words)
def print_first_and_last(sentence):

        words =break_words(sentence)
        print_first_word(words)
        print_last_word(words)

def print_first_and_last_sorted(sentence):
        words=sort_sentence(sentence)
        print_first_word(words)
        print_last_word(words)
```

Now, we import and run as follows...

```
>>> import ex25
>>> sentence="good things come to those who wait"
>>> words=ex25.break_words(sentence)
>>> words
['good', 'things', 'come', 'to', 'those', 'who', 'wait']
>>> sorted_words = ex25.sort_words(words)
>>> sorted_words
['come', 'good', 'things', 'those', 'to', 'wait', 'who']
>>> ex25.print_first_word(words)
good
>>> ex25.print_last_word(words)
wait
>>> ex25.print_first_word(sorted_words)
come
>>> ex25.print_last_word(sorted_words)
who
>>> sorted_words
['good', 'things', 'those', 'to', 'wait']
>>> sorted_words = ex25.sort_sentence(sentence)
>>> sorted_words
['come', 'good', 'things', 'those', 'to', 'wait', 'who']
>>> ex25.print_first_and_last(sentence)
good
wait
>>> ex25.print_first_and_last_sorted(sentence)
come
who
```

Here, successive function calls modify *words* and *sorted_words* . Those modified variables get passed to the next function call.