Functions and Methods

Introduction to Data Science with Python

Functions

- We have already seen and used functions provided by Python
  - String functions (len, print, lower)
  - Numeric functions (abs, sqrt, log)
  - List, tuple, dictionary functions
- There are many more and we can also create our own

Dates and Time

- Let's start with date and time functions

  Base Python uses the datetime module

  Import: date, time, and datetime (remember how to do this?)

```
# Date Time
from datetime import date, time, datetime
print(date(2014,7,23))

d1 = date(2014,7,23)

d2 = date(2014,7,28)

d1 - d2

print(datetime(2014,7,28,12,58,24))
datetime(2014,7,28,12,58,24).date()
datetime(2014,7,28,12,58,24).weekday()
```

Formatting / Parsing

- Method to convert from string to date object

  Use datetime.strptime()

- Method to convert from Date to String

  Use datetime.strftime()

```
# formatting
dt2 = datetime.strptime("2014/07/11","%Y/%m/%d")
print(dt2)
>> 2014-07-11 00:00:00
dt2.strftime("%Y-%m")

>> '2014-07‘


help(datetime)
```

Test Your Knowledge – Ex 6.1

- Exercise – your turn

1. Convert your birthday to a date
2. Print the date your were born
3. Print the weekday you were born

String Methods

- Review some sample string methods

```
# String methods
s1 = "First part, "
s2 = "Second part."
print(s1 + s2)
>> First part, Second part.
print(s1.upper())
>> FIRST PART,
print(s1.replace("First","Gone").lower())
>> Gone part,
print(s1.count("r"))
>> 2
help(str)
```

Numeric Functions/Methods

- Review some sample Numeric functions

```
# Sample Math Functions
x / y
x * y
x + y
x - y
-x
abs(-x)
x**2


import math
math.sqrt(y)
```

User-Defined Functions

- Sometimes you just have to “roll your own”
- Use the def keyword to create your own user-defined functions

```
# create a function
def sayHello(name):
    print("Hello " + name)

sayHello("Rupert")

>>> Hello Rupert
```

User-Defined Functions

```
def new_func(par1,par2,some_other_par="default:"):
    do some stuff
    return some_related_stuff
```

```
# Sample Function
def mult_by_3(x,weight=1):
    result = x*weight*3
    return result
mult_by_3(4)
>> 12
mult_by_3(4,0.5)
>> 6.0
mult_by_3(weight=0.25,x=4)
>> 3.0
result #error
```

Test Your Knowledge – Ex 6.2

- Now it's your turn to apply what you have learned.

1. Create a list of five elements from the numbers 1 through 5
2. Create a function to take a list and a value and multiply each element of the list by the value, then return the resulting list
3. Try your function with your list and the value 10. Print the result.

Avoiding Keywords/Functions

- Python keywords cannot be overwritten
- Python built-in functions can be overwritten, but is a bad idea…

```
# Keywords
import keyword
keyword.kwlist

['and',
 'as',
 'assert',
 'break',
 'class',

…
```

Built-in Functions

From Official Python Docs

abs()	divmod()	input()	open()	staticmethod()
all()	enumerate()	int()	ord()	str()
any()	eval()	isinstance()	pow()	sum()
basestring()	execfile()	issubclass()	print()	super()
bin()	file()	iter()	property()	tuple()
bool()	filter()	len()	range()	type()
bytearray()	float()	list()	raw_input()	unichr()
callable()	format()	locals()	reduce()	unicode()
chr()	frozenset()	long()	reload()	vars()
classmethod()	getattr()	map()	repr()	xrange()
cmp()	globals()	max()	reversed()	zip()
compile()	hasattr()	memoryview()	round()	__import__()
complex()	hash()	min()	set()	apply()
delattr()	help()	next()	setattr()	buffer()
dict()	hex()	object()	slice()	coerce()
dir()	id()	oct()	sorted()	intern()

Functions vs Methods

- Built-in Functions are stand-alone objects

  Some_function(arguments)

- Functions Attached to Objects are called Methods

  Object.function(argument)

```
# Functions / Methods
var1 = "foo"

# Function
print(len(var1))
>> 3

# Method
print(var1.capitalize())
>> Foo
```

Functions vs Methods

- Objects have attached functions (methods) and other attributes

  Object.__attribute__()

- Spyder/IPython shows the objects methods/attributes
- Use the Object Inspector in Spyder to see these methods and attributes

```
# attribute
print(var1.__len__())

>> 3
```

Test Your Knowledge – Ex 6.3

- Write a function that removes line end characters (line feed and carriage return/line feed) from a string.

Variable Scope

- Variable in a function are defined only in the "local" namespace
- Global variables are not affected/touched by default

```
# Global var function
def mult_by_3(x,weight=1):
    global result
    result = x*weight*3
    return result
mult_by_3(4)
>> 12
result
>> 12
```

Namespace

- The global statement notifies the function the variable is in the global namespace.

```
# global 2
my_global_var = 12
def mult_by_3(x,weight=1):
    global my_global_var
    result = x*weight*3*my_global_var
    return result
mult_by_3(4)
>> 144
```

Function Output

- Use the return function to pass output from a user-defined function
- Assign the output to a variable just like any other value

```
def fullName(firstName, lastName):
    return(firstName + ' ' + lastName)


fullName('Rupert', 'Jones’)


>>> 'Rupert Jones'
```

Test Your Knowledge – Ex 6.4

- Write a function that takes as parameters the start and stop year. Have the function print out all the names of the first days of each month for each year listed

Unpacking Output

- You can combine variables into a tuple and return the tuple
- Python will automatically unpack it if required

```
def namesFromFullName(fullName):
    # this returns a list
    names = fullName.split(' ')
    return(names)


# firstName gets the first element, lastName gets the
# second element
firstName, lastName = namesFromFullName('Rupert Jones')
```

Functions (multiple values)

- Functions can return multiple values

  And also unpack them!

```
# Multiple Values
def multi_func(a,b,c):
    return (a + 1,b + 3,c - 3)

a1, b1, c1 = multi_func(1,1,1)

a1
>>  2
```

Multiple Return Statements

- Function can have multiple return statements
  - Usually used in conditional returns

```
# Multiple Return Statements
def multi_func2(a):
    if a > 0:
        return a + 1
    if a < 0:
        return a - 1
multi_func2(5)
>> 6
multi_func2(-5)
>> -6
```

Anonymous Functions

- So called "lamba" functions

  One line function expressions (useful construct in data science)

  new_func = lambda *par [,par2,…]: expression

```
# Lambda
lam_func = lambda x1,x2: x1 + x2
lam_func(3,4)


# Multiple returns
lam_func2 = lambda x1,x2: (x1 + x2,x1 - x2)
lam_func2(3,4)
```

Test Your Knowledge – Ex 6.5

- The Fibonacci sequence starts with 0 and runs like this:

0, 1, 1, 2, 3…

- The formula (n > 2) is this

$X_n = X_{n-2} + X_{n-1}$

- Create a function to generate a Fibonacci sequence whose final value is less than or equal to a given value
- Can you do this as a lambda function?

*args and **kwargs

- We know we can parameters to functions, but what if we don't know what or how many of those parameters we are supposed to pass?
  - You could try to bundle them in a list, but Python offers a cleaner way via variable parameters *args and **kwargs
- *args and **kwargs are for passing multiple arguments to a function
  - The names are conventions, you can call them anything you want, just don't forget the asterisk(s)
- So what's the difference?
  - Use *args for a non-keyworded variable length argument list
  - Use **kwargs for a keyworded variable length argument list (i.e. a dictionary)
    - args = argument list, kw = keyworded, get it?

* args Example

- To use the *args parameter, declare it, then iterate without the asterisk using a for loop or other iterator

```
# using *args
def test_args(reg_arg, *args):
    print("regular arg: ", reg_arg)
    for arg in args:
        print(arg)

test_args("Good-bye ", "cruel ", "world")
```

regular arg: Good-bye
cruel
world

**kwargs Example

- Note here we are passing named parameters via a pseudo-dictionary
- We can bundle the variables into a real dictionary and pass that too

```
# using *kwargs
def test_kwargs(**kwargs):
    if kwargs is not None:
        for key, value in kwargs.items():
            print("{} == {}".format(key, value))

test_kwargs(var1 = "Good-bye ", var2 = "cruel ", var3 = "world")
var1 == Good-bye
var2 == cruel
var3 == world
```

Lab 6 (60 min)

- Functions and the Three letter game - Part 1

```
Take the Lab 6 - Start.py file
   In it you have some starter code to build a game
   Fill in the function definitions
   If you have time, create some test code
```