EASC2410 Lecture 4

# Python Basics: Functions and Modules

Dr. Binzheng Zhang
Department of Earth Sciences

# Review of Lecture 3

**In Lecture 3, we learned:**

- **Dictionaries**
- **Relational and logical operations**
- **Basic Python program structures**
- **if, if-else, if-elif-else statements**
- **while loops**
- **for loops**
- **nested loops**

**In Lecture 4, you will learn:**

- **Define and call Functions**
- **Modules**

# Functions in Python

**What are functions?**

Functions are a bunch of Python code that are re-used in complicated programs

**Why we need functions?**

Functions make your code more efficient, readable and re-usable

**How to create functions in Python?**

Use the **`def`** statement (and **`return`**, which is optional)

**Here is a simple example of creating (defining) a Python function**

```
def convertF2C(in_args):
    """
    This code convert Fahrenheit (F) to Celsius (C)
    INPUT    : temperature in F
    OUTPUT   : temperature in C
    Algorithm: C = 9/5*(F-32)
    """
    out_args = (in_args - 32.0)*5.0/9.0
    return out_args
```

**How to use (call) the function in your Python code?**

Just type in the function name with appropriate ***input arguments***

```
convertF2C(100)
```

```
Out[19]: 37.77777777777778
```

# How does the code work?

```
1  def convertF2C(in_args):
2      """
3      This code convert Fahrenheit (F) to Celsius (C)
4      INPUT    : temperature in F
5      OUTPUT   : temperature in C
6      Algorithm: C = 9/5*(F-32)
7      """
8      out_args = (in_args - 32.0)*5.0/9.0
9      return out_args
10
11 # now lets call the function here by simply type in the function name
12 convertF2C(100)
```

Line 1: `def convertF2C(in_args)`

- `def` the first line of a function must have def as the first three letters, it tells Python that you're **defining** a function
- `convertF2C` the **name** of the function
- `(in_args)`: **input** arguments
- `:` the first line of a function always **ends** with a terminal colon (DON'T FORGET IT!)

Line 2-7: """

This code convert Fahrenheit (F) to Celsius (C)
INPUT: temperature in F
OUTPUT: temperature in C
"""

- triple quotes right after the function definition is a **doc string**
- The doc string is a description of what the function does, which can be accessed later by using the help() function.

# How does the code work?

```
1  def convertF2C(in_args):
2      """
3      This code convert Fahrenheit (F) to Celsius (C)
4      INPUT    : temperature in F
5      OUTPUT   : temperature in C
6      Algorithm: C = 9/5*(F-32)
7      """
8      out_args = (in_args - 32.0)*5.0/9.0
9      return out_args
10
11 # now lets call the function here by simply type in the function name
12 convertF2C(100)
```

Line 8: `out_args =(in_args - 32.0)*5.0/9.0`

- The body of the **convertF2C** function
- First calculates the temperature using **in_args** through the formula C = 9/5*(F-32)
- Then assign the calculated value to a variable named **out_args**

Line 9: `return out_args`

- **returns** the results of whatever the function did.
- Here it returns the value of temperature in C (that's why you get the output)

Line 12: `convertF2C(100)`

- **Call** the function you just defined above and give it an input argument of 100 degrees
- Now the function will convert 100 degrees to radians and return the results to screen
- without the `return` statement, there's no value get back after the function is called (you get nothing)

# Tips on Defining a function

## The doc string

The Doc String briefly describes what the code does; weeks after you've written your code, it will remind you of what you did. In addition, you can use it to print out a help message that lets others know what the program does.

The **help()** function will print out the doc string in your function, for example:

In [2]:
```
1 help(convertF2C)
```

```
Help on function convertF2C in module __main__:

convertF2C(in_args)
This code convert Fahrenheit (F) to Celsius (C)
INPUT    : temperature in F
OUTPUT   : temperature in C
Algorithm: C = 9/5*(F-32)
```

## The function body

This part of the code MUST be ***indented***, just like in a **for** loop, or you can just regard the function body as a block of Python code under the **`def`** statement.

## The return statement

Python separates the input and output arguments. Incoming arguments are *passed through* the **`def`** statement and returning arguments get *shipped out* with the **`return`** statement.

# Tips on Defining a function

## The scope of a variable - “local” variables versus “global variables”

Inside a function, variable names have their own meaning which in many cases will be different from outside the calling function. In other words, variable names declared inside a function stay in the function and cannot be accessed outside the function.

```
def LasVegas():
    """ This is a test function
    """
    V='Casinos!' # assign a string to V
    return       # V only valid inside LasVegas()

LasVegas() # call LasVegas()
print (V)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-20-5bb8632b1a7f> in <module>()
      6
      7 LasVegas()
----> 8 print (V)

NameError: name 'V' is not defined
```

In this case, `V` is called a “**local**” variable, which is only valid inside the function `LasVegas()`. So the variable V only exists when the function `LasVegas()` is called. After the function call, V is “destroyed” by Python, that’s why when the print() function tried to access variable V, it gave an error message saying “name ‘V’ is not defined” - quite easy to understand

```
def SanDiego():
    """ This is another test function
    """
    global G     # now G is a global variable
    G='Surfing!' # assign a string to V
    return # G valid inside and outside SanDiego()

SanDiego() # call SanDiego()
print (G)
```

```
Surfing!
```

In this case, `G` is called a “**global**” variable, which is not only valid inside the function `SanDiego()`, but also outside the function `SanDiego()`. So the variable G is created when the function `SanDiego()` is called. After the function call, G is “kept” by Python, that’s why when the print() function tried to access variable V, it gave ‘Surfing!’ - very SanDiego-ish

# Passing arguments to functions

## Pass fix amount of arguments: `func_name(arg1, arg2, arg3,…)`

```
def deg2rad(degrees):
    """
    This code converts degrees to radians
    INPUT:  degree
    OUTPUT: radians
    """
    return degrees*3.1415926535897793/180.

# now let's use the function with input  = 40 degrees
print ('42 degrees in radians is',deg2rad(40))
```

```
42 degrees in radians is 0.6981317007977318
```

You can also pass arguments with **default** values specified using the “=” sign

```
def ask_ok(retries=2, reminder='Please try again!'):

    while True:
        ok = input('Please enter yes or no: ')
        if ok in ('y', 'ye', 'yes', 'yup'):
            return True
        if ok in ('n', 'no', 'nop', 'nope'):
            return False
        retries = retries - 1
        if retries <= 0:
            print('too many tries...invalid user response, exiting!')
            return

# let's try call the function ask_ok() without arguments
# in this case, Python will use the default values as specified in the function definition
# with retires = 2 and reminder = 'Please try again!'
ask_ok()
```

```
Please enter yes or no: yes

True
```

# Passing arguments to functions

## Pass a variable amount of arguments - *variadic functions:* `func_name(*arg)`

```
def print_args(*args):
    """
    prints argument list
    """
    print (type(args))  # args is a tuple that you can step (like a list)
    print ('You sent me these arguments: ')
    for element in args: # step through all the elements in the input argument
        print (element)  # print each element in the input argument

# now let's try call the print_args function with different
print_args(42, True, [1,4,'hi there'])
```

```
<class 'tuple'>
You sent me these arguments:
42
True
[1, 4, 'hi there']
```

This function takes a variable amount of input (a tuple) and prints out all the elements in the input

## Pass arguments as key-value pairs: `func_name(**arg)`

Another way is to use any number of so-called *keyword-value* pairs. This is done by putting double * (e.g., **args) as the last argument in the argument list. kwargs stands for key word arguments and is treated like a list in the function. This is probably not gonna be used a lot throughout this course, so we are not going to talk too much about it. Here's more information from the Python documentation/tutorial: https://docs.python.org/3/tutorial/controlflow.html#keyword-arguments

# The `main()` program

It is usually considered as good Python style to treat your main program block as a function too. (This helps with using the "**doc string**" as a help function and building good program documentation in a large project). The following example shows how to define and use a `main()` function

```
# fist define all the functions that's needed in your python main program
def deg2rad(degrees):
    """
    converts degrees to radians
    """
    return degrees*3.141592653589793/180.

def convertF2C(in_args):
    """
    This code convert Fahrenheit (F) to Celsius (C)
    INPUT    : temperature in F
    OUTPUT   : temperature in C
    Algorithm: C = 9/5*(F-32)
    """
    out_args = (in_args - 32.0)*5.0/9.0
    return out_args

# then put all the codes together in the main() program
def main():

    my_degree     = 75.0
    my_fahrenheit = 75.0

    my_radian  = deg2rad(my_degree)
    my_celsius = convertF2C(my_fahrenheit)

    print(my_degree,'degrees in radians is',my_radian)
    print(my_fahrenheit,'F in Celsius is',my_celsius)

# now run the main program, it's in the last line of your code without indentation
main()
```

```
75.0 degrees in radians is 1.3089969389957472
75.0 F in Celsius is 23.88888888888889
```

# Modules in Python

**What are Python modules?**

A collection of functions that can be used in any Python codes

**Why we need modules?**

Efficient and make your function re-usable, or save your time by using other people's modules

**How to create your own modules in Python?**

Type in all your functions in a python script with suffix .py, and save it somewhere Python can access

**How to use modules in Python?**

Use the `import` command:

```
import MODULE
import MODULE as MODULE_NICKNAME
from MODULE import SUB_MODULE
from MODULE import *
```

**Data science in Python is all about modules!**

# Create a module in Jupyter

## Step 1. Open the Jupyter dashboard by clicking the jupyter symbol


## Step 2. Create a new txt file by clicking “New” and then select “Text File”

## Step 3. Type in your functions in the .txt file

jupyter untitled.txt a minute ago

Logout

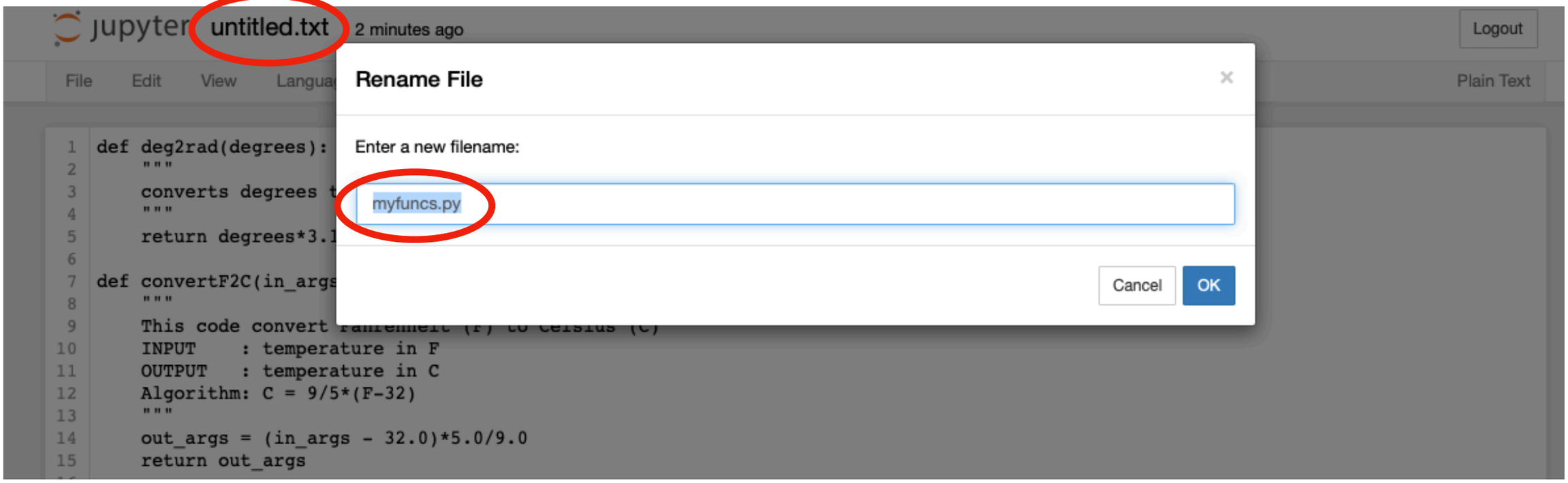
File Edit View Language Plain Text

```
def deg2rad(degrees):
    """
    converts degrees to radians
    """
    return degrees*3.141592653589793/180.

def convertF2C(in_args):
    """
    This code convert Fahrenheit (F) to Celsius (C)
    INPUT    : temperature in F
    OUTPUT   : temperature in C
    Algorithm: C = 9/5*(F-32)
    """
    out_args = (in_args - 32.0)*5.0/9.0
    return out_args

def SanDiego():
    global G
    G='Surfing!'
```

## Step 4. Rename the .txt file to be myfuncs.py by clicking the name “untitled.txt”



# Use a module in Jupyter

Now you have created your first Python module called **myfuncs.py**. To use those functions in your module, simply use the **`import`** command:

```
import myfuncs # import your own module calld myfuncs (that's the .py script you just saved to your directory)
               # this module has three functions, deg2rad(), convertF2C(), SanDiego()

print(myfuncs.deg2rad(45))    # call the function deg2rad and print the results
print(myfuncs.convertF2C(45)) # call the function convertF2C and print the results
```

```
0.7853981633974483
7.222222222222222
```

Recall the `help()` function, which is also applicable to modules in Python:

```
help(myfuncs)
```

```
Help on module myfuncs:

NAME
    myfuncs

FUNCTIONS
    SanDiego()

    convertF2C(in_args)
        This code convert Fahrenheit (F) to Celsius (C)
        INPUT    : temperature in F
        OUTPUT   : temperature in C
        Algorithm: C = 9/5*(F-32)

    deg2rad(degrees)
        converts degrees to radians

FILE
    /Users/bzhang/Dropbox/Teaching/Python for Earth Sciences/Notebooks/myfuncs.py
```

# Useful modules in Python

### *Standard Modules¶*

- the **os** module: provides functions for interacting with the operating system
- the **sys** module: provides access to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter
- the **math** module: provides access to the mathematical functions defined by the C standard.
- the **datetime** module: supplies classes for manipulating dates and times in both simple and complex ways.

### *Numerical Analysis Modules¶*

- the **numpy** module: the fundamental package for scientific computing and analysis with Python.
- the **scipy** module: a collection of numerical algorithms and domain-specific toolboxes, including signal processing, optimization, statistics and much more

### *Data handling Modules¶*

- the **pandas** module: an open source library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language

### *Visualisation Modules¶*

- the **matplotlib** module: a mature and popular plotting package, that provides publication-quality 2D plotting as well as rudimentary 3D plotting

**numpy + scipy + matplotlib** basically give you Matlab

**numpy + pandas + matplotlib** = basic data analysis toolkit

## For example, use the `math` module

Load the math module, use: **`import`** `math`

access data in math module, use: `math.pi, math.e, math.tau`

call functions in math module, use: `math.cos(), math.log()` …

```
import math # import the "math" module, and use "mt" as a short name

print(math.pi)                  # give me pi
print(math.cos(math.pi/5.0))    # cosine function
print(math.log(1024,2))         # logrithm function
```

```
3.141592653589793
0.8090169943749475
10.0
```

Rename your math module, use: **`import`** `math` **`as`** `mt`

Or, if you don't want to use a prefix for the functions in the math module, use:

**`from`** `math` **`import`** `*`

```
from math import *    # import the "math" module, and use no prefix
print(pi)             # give me pi
print(cos(pi/5.0))    # cosine function
print(log(1024,2))
```

```
3.141592653589793
0.8090169943749475
10.0
```