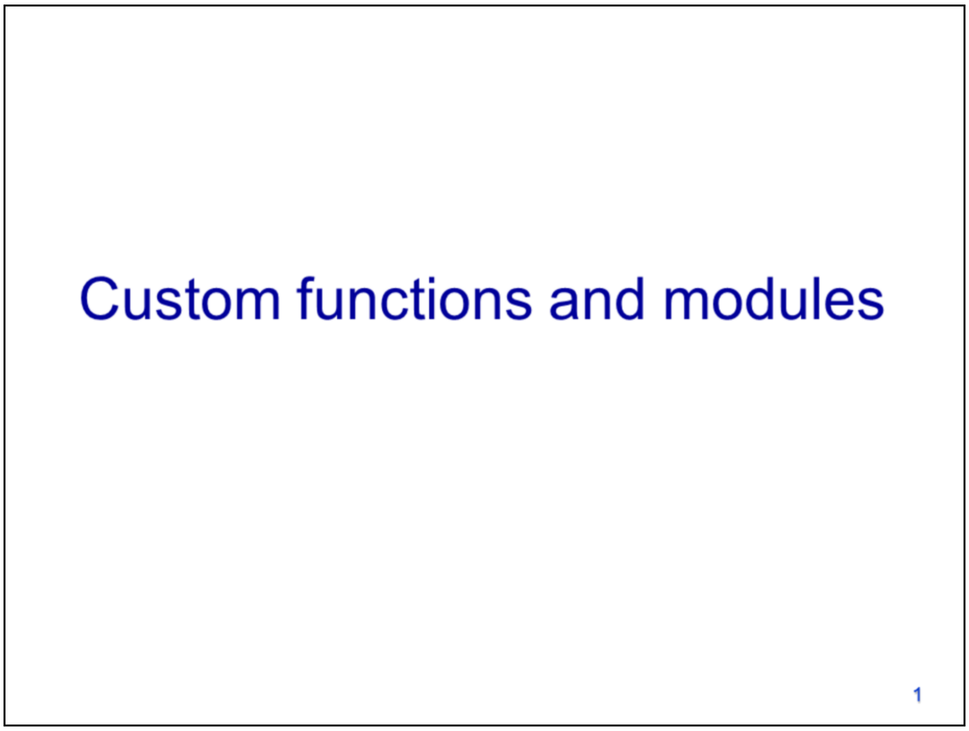
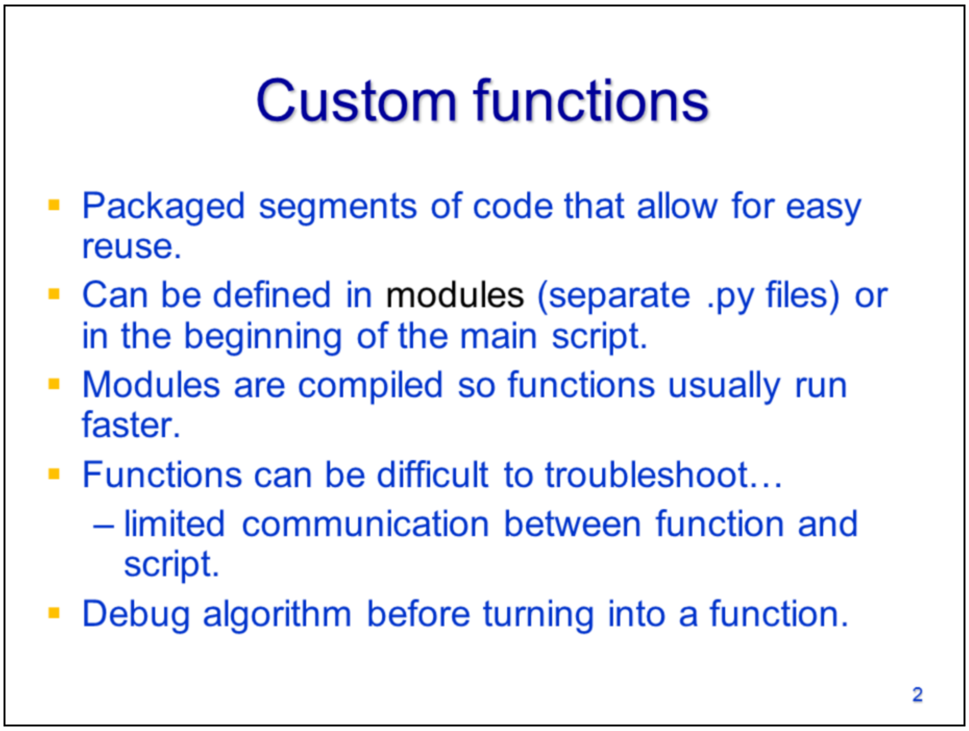


# Custom functions and modules

1

This video will discuss how to create custom functions and modules.



## Custom functions

- Packaged segments of code that allow for easy reuse.
- Can be defined in modules (separate .py files) or in the beginning of the main script.
- Modules are compiled so functions usually run faster.
- Functions can be difficult to troubleshoot…
  - – limited communication between function and script.
- Debug algorithm before turning into a function.

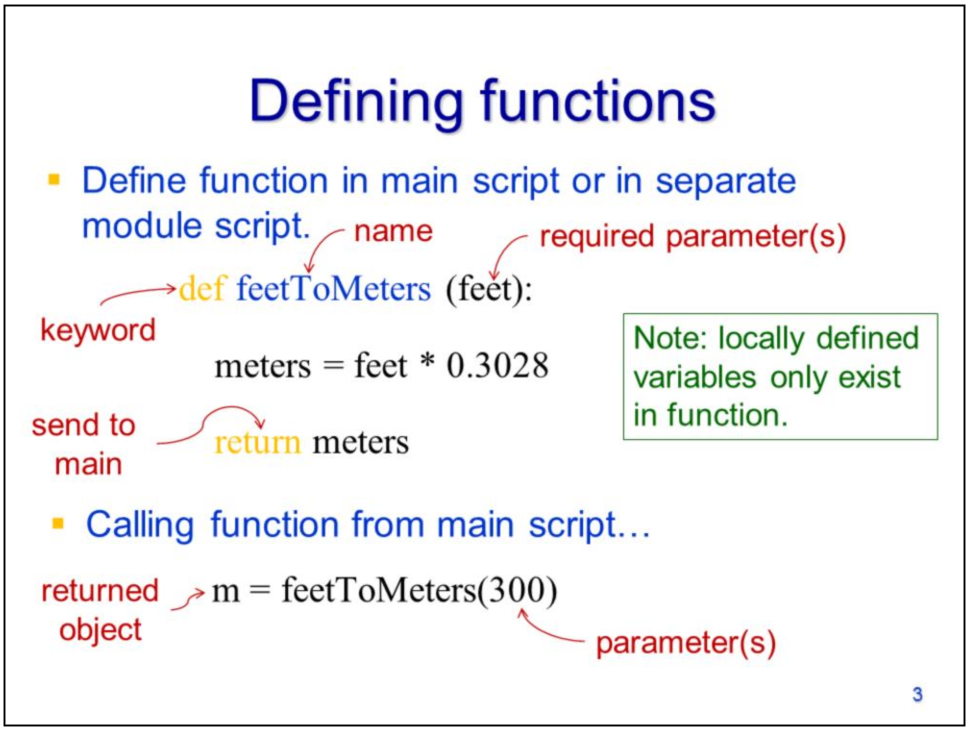
2

Functions are tools written to perform specific tasks which are packaged to allow for easy reuse in future scripts. Functions also help to organize a script and make it easier to upgrade.

Modules are script files that contain functions and may be called by other scripts. Functions can also be defined in the same script that calls it – in these cases, the function should be defined at the beginning of a script before it is called.

Module scripts have the advantage of being easily accessible by multiple scripts. Functions in modules also run faster than normal scripts because they are compiled.

Code should be thoroughly tested before converting it into a function because it is more difficult to troubleshoot code after it has been converted to a function.



## Defining functions

- Define function in main script or in separate module script.

```
def feetToMeters (feet):

    meters = feet * 0.3028

    return meters
```

name
required parameter(s)
keyword
send to main

Note: locally defined variables only exist in function.

- Calling function from main script…

```
m = feetToMeters(300)
```

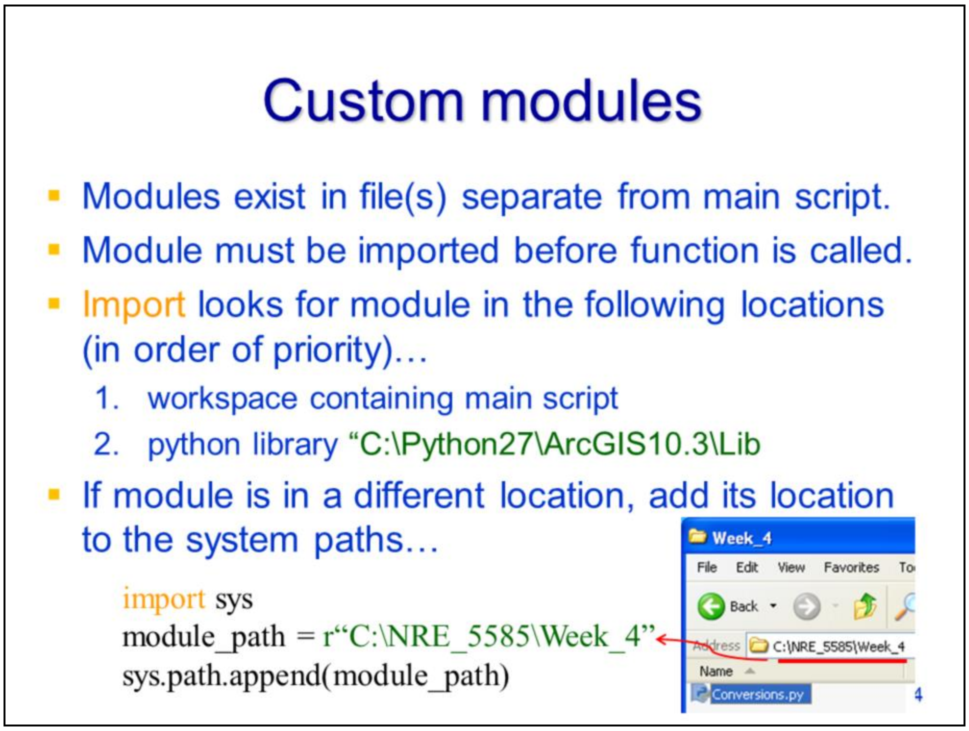
returned object
parameter(s)
3

The method for defining a function is the same regardless of whether it is defined in a separate module script or in the main script which calls (i.e. runs) the function.

Functions are compound statements. The header line begins with the **def** keyword followed by the name that will be assigned to the function.

Any parameters are indicated after the function name. Note that variables defined within a function are not accessible to the script that calls the function. The **return** statement sends one or more objects to the main script.

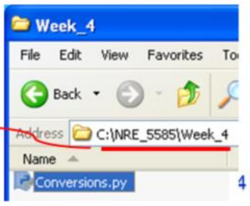
The syntax for calling a function differs depending on whether the function is defined in a separate module script or the main script. To call a function that was defined in the main script, 1) specify the function name followed by the values that will be assigned to the function's parameters. If the function has a return statement, then you will need to assign a variable to the returned object.



## Custom modules

- Modules exist in file(s) separate from main script.
- Module must be imported before function is called.
- Import looks for module in the following locations (in order of priority)…
  1. workspace containing main script
  2. python library "C:\Python27\ArcGIS10.3\Lib
- If module is in a different location, add its location to the system paths…

```
import sys
module_path = r"C:\NRE_5585\Week_4"
sys.path.append(module_path)
```



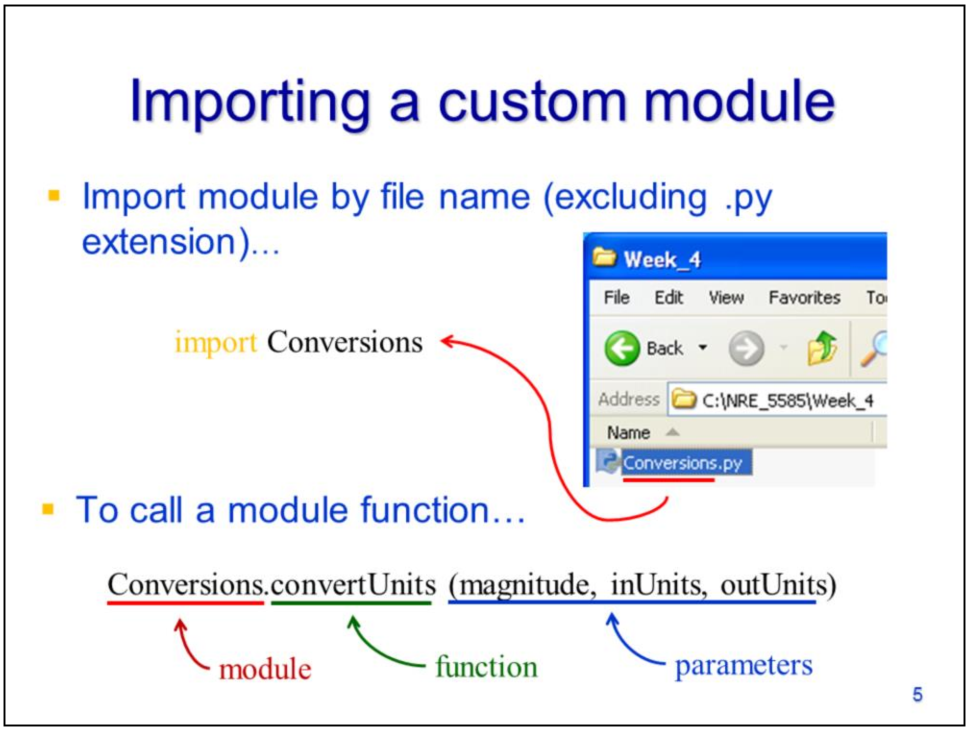
4

Functions defined in a module script exist in a different file from the main script that is calling the function.

In order for the main script to use a module, it first needs to import it.

In order to import a module, the script needs to be able to find it. The main script will first look in the workspace that contains the main script. If the module script is not found, then the main script will look in the python library folder. When a module is likely to be used by many different scripts, it will be most efficient to save the module script to the python library folder. If the module is being used by a script that you intend to share, then it will be more convenient to copy the module script to the folder containing the main script.

You can also direct the main script to look in other folders by appending the folder location to **sys.path's** append method. This will need to be done prior to importing the module.



## Importing a custom module

- Import module by file name (excluding .py extension)...

Week_4
File Edit View Favorites To
Back
Address C:\NRE_5585\Week_4
Name
Conversions.py

```
import Conversions
```

- To call a module function...

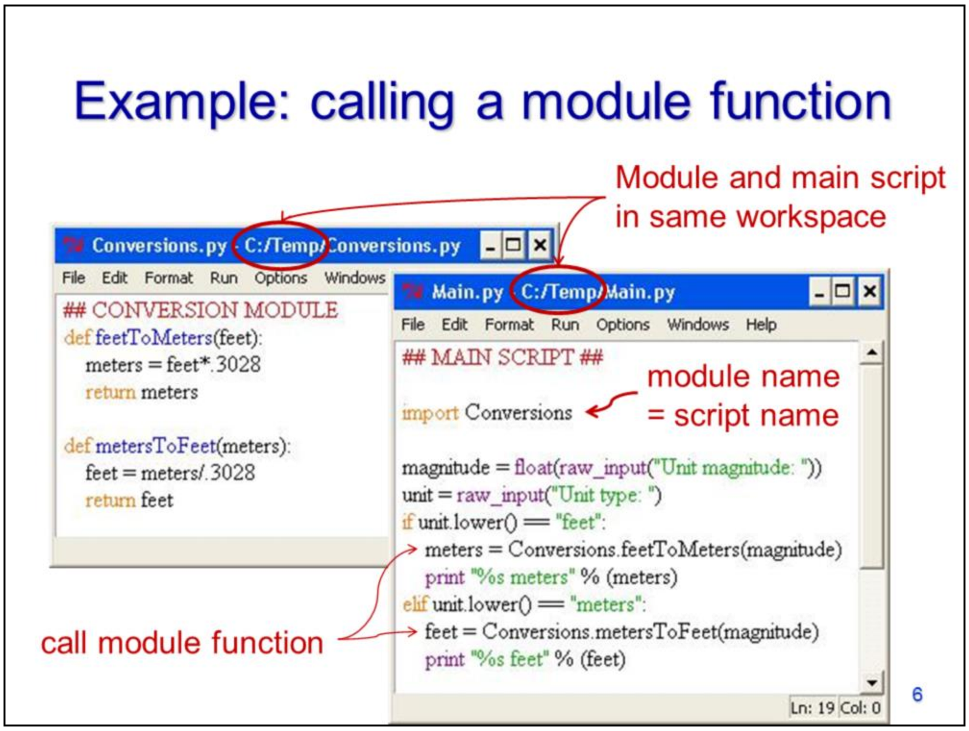
```
Conversions.convertUnits (magnitude, inUnits, outUnits)
```

module
function
parameters
5

Import a module by specifying the name of the script file with the extension excluded from the name.

Call a function from a module using the same syntax that we've used previously for the built-in modules:

Specify the module name, followed by the function name, followed by the parameter values.


Conversions.py - C:/Temp/Conversions.py
File Edit Format Run Options Windows
Main.py - C:/Temp/Main.py
File Edit Format Run Options Windows Help
Module and main script in same workspace
module name = script name
call module function
Ln: 19 Col: 0
6

## Example: calling a module function

```
## CONVERSION MODULE
def feetToMeters(feet):
    meters = feet*.3028
    return meters

def metersToFeet(meters):
    feet = meters/.3028
    return feet
```

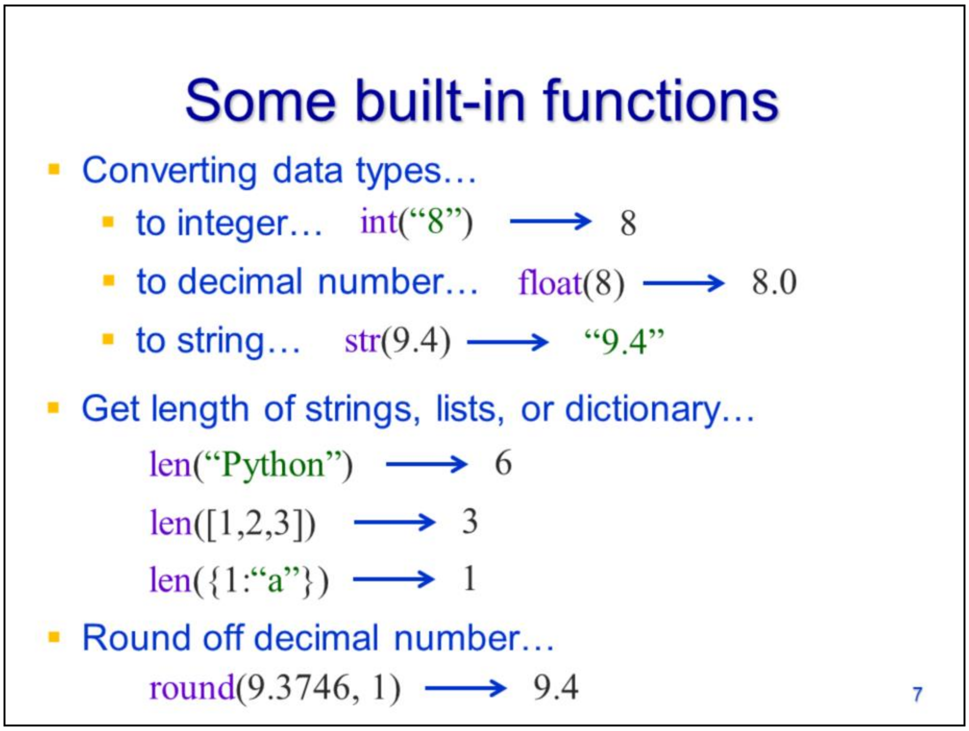
```
## MAIN SCRIPT ##

import Conversions

magnitude = float(raw_input("Unit magnitude: "))
unit = raw_input("Unit type: ")
if unit.lower() == "feet":
    meters = Conversions.feetToMeters(magnitude)
    print "%s meters" % (meters)
elif unit.lower() == "meters":
    feet = Conversions.metersToFeet(magnitude)
    print "%s feet" % (feet)
```

Let's look at an example of how to call a module function from the main script. This is the module script containing a set of functions.

Here is the main script. Note that the module script is located in the same folder as the main script.

Import the module into the main script using the file name for the module script.

Call functions from the module script by specifying the module name, the function name, and the function parameters.



## Some built-in functions

- Converting data types…
  - to integer… int(“8”) $\longrightarrow$ 8
  - to decimal number… float(8) $\longrightarrow$ 8.0
  - to string… str(9.4) $\longrightarrow$ “9.4”
- Get length of strings, lists, or dictionary…
  - len(“Python”) $\longrightarrow$ 6
  - len([1,2,3]) $\longrightarrow$ 3
  - len({1:“a”}) $\longrightarrow$ 1
- Round off decimal number…
  - round(9.3746, 1) $\longrightarrow$ 9.4

7

Here we’ll look at some common functions in the built-in module – this module loads automatically in Python and does not need to be imported.

The **int** function converts strings or decimal numbers to an integer format. A string must contain only a single number with no decimal. The string can have spaces before or after the number. Note that when a decimal number is converted to an integer, the decimal part is simply truncated.

The **float** function converts strings or integers to a decimal number.

The **str** function converts any number type to a string.

The **len** function gets the number of characters in a string, the number of items in a lists, or the number of entries in a dictionary.

The **round** function rounds off a decimal number to a specified number of digits after the decimal.