

# Lab 2: Programming with Python

*Ben Best*

*October 10, 2014*

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Data</b>	<b>2</b>
<b>3</b>	<b>Methods</b>	<b>3</b>
3.1	Download Data . . . . .	3
3.2	Open RStudio (and configure git.exe path again) . . . . .	3
3.3	Inspect and Run Lab 1 Model . . . . .	4
3.3.1	Inspect Sub-Models . . . . .	5
3.3.2	Slight Differences in Model . . . . .	5
3.4	Export Models to Scripts . . . . .	6
3.5	Open Script in WinPython . . . . .	6
3.6	Anatomy of an Arcpy Script . . . . .	7
3.7	Intro to Python . . . . .	7
3.7.1	String Formatting and Variable Substitution . . . . .	8
3.7.2	Module Import, Whitespace, For Loop, Zero-Based Indexing . . . . .	8
3.8	Python Window in ArcMap . . . . .	9
3.9	Add Loop in Pythonwin . . . . .	10
3.10	Add Summary Functions to Script . . . . .	14
3.11	Change to EEZ data . . . . .	15
3.12	Turn in Report . . . . .	16
<b>4</b>	<b>Review</b>	<b>16</b>

## 1 Introduction

The United Nations has tasked you to provide a summary of offshore wind data (versus terrestrial last time). Once again they want the wind speed data summarized per country and across months of the year. They also want the results as a script to easily reproduce and fully document the parameters for this analysis.

### Learning Goals

You'll work from a familiar Model Builder analysis to produce and understand a Python script. You'll only be changing the input country data (offshore EEZ vs terrestrial) to produce a different result. The point here is to provide you more time to replicate the analysis in a new programming environment.

## 2 Data

The data is the same as last week's lab, plus the offshore Exclusive Economic Zones (EEZ).

- **NCDC.NOAA.gov: NCEP Reanalysis Wind Data** <http://www.ncdc.noaa.gov/societal-impacts/wind>
  - spatial resolution: 2.5 x 2.5 degrees
  - temporal resolution: daily
  - format: netcdf (\*.nc)
  - files: H:\esm296-4f\labs\lab2\raw\uwnd.sig995.2013.nc, vwnd.sig995.2013.nc

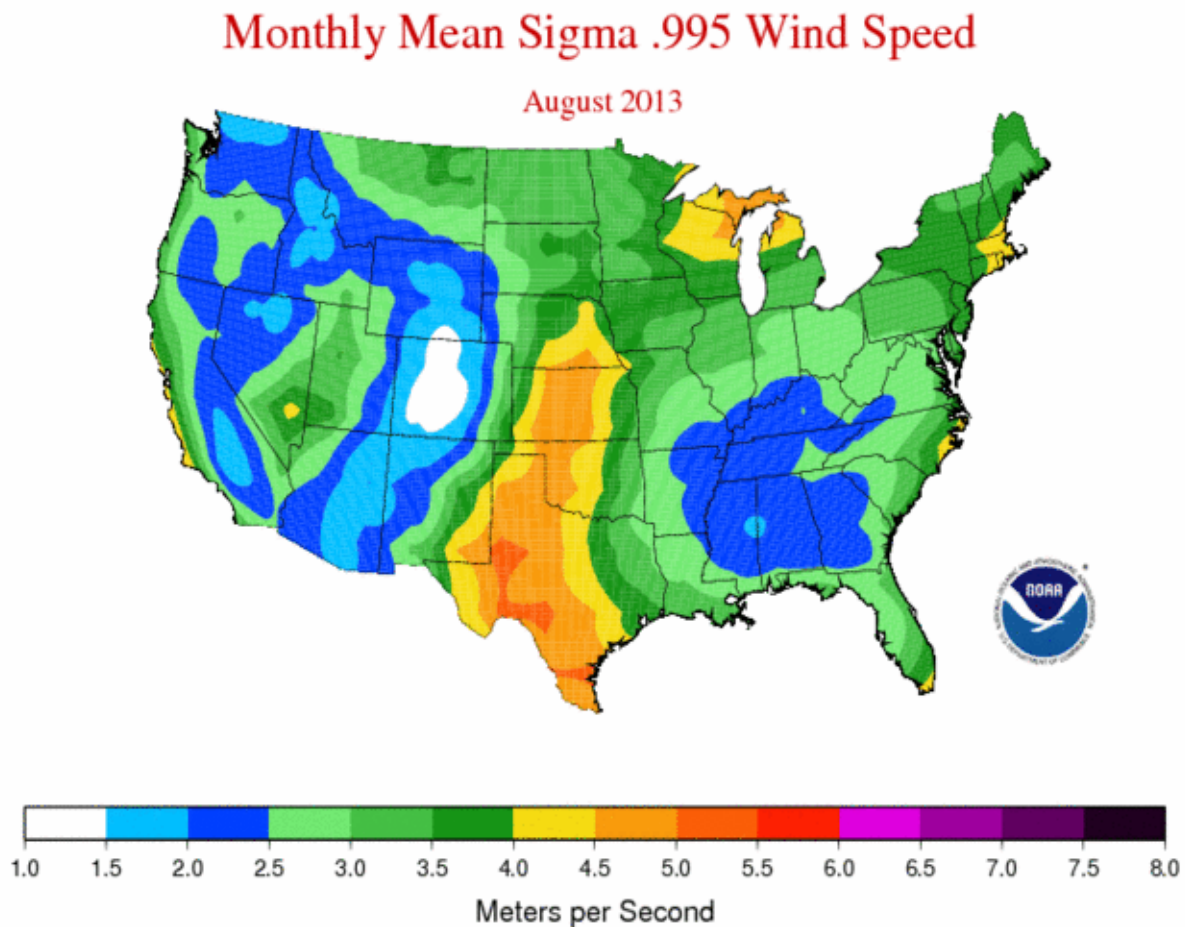


Figure 1:

- **NaturalEarthData.com: 1:10m Cultural Vectors - Admin 0 – Countries** <http://www.naturalearthdata.com/downloads/10m-cultural-vectors/10m-admin-0-countries/>
  - spatial resolution: 1:10m
  - version: 3.1.0
  - format: shapefile (\*.shp)

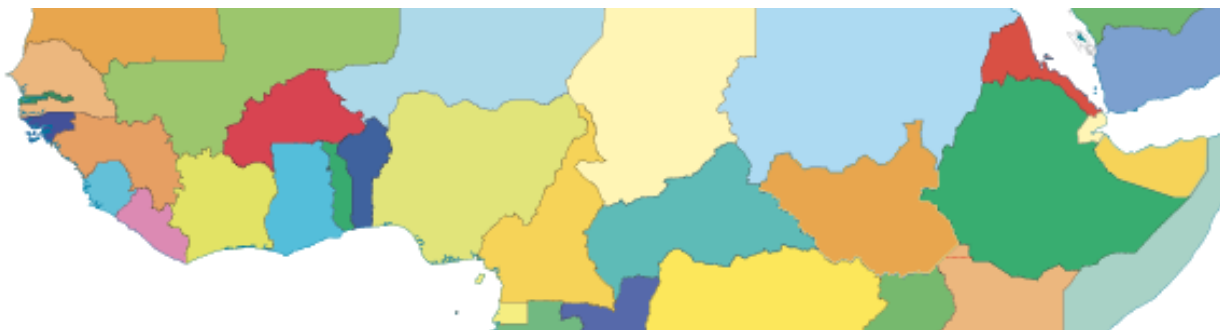


Figure 2:

- file: H:\esm296-4f\labs\lab2\raw\ne\_10m\_admin\_0\_countries.shp
- **MarineRegions.org: Exclusive Economic Zones Boundaries (EEZ)** <http://www.marineregions.org/sources.php#eez>
  - version: Low res v8 (2014-02-28, [history](#), [issues](#), [methodology](#))
  - format: shapefile (\*.shp)
  - file: H:\esm296-4f\labs\lab2\raw\World\_EEZ\_v8\_2014.shp



Figure 3:

The data this week is bundled into a zip file for you, the download of which is detailed for you below.

## 3 Methods

### 3.1 Download Data

You should've already reorganized your file system. Download the preparatory lab2 files here:

[lab2.7z](#)

I recommend placing this in H:\esm296-4f\labs, right clicking 7-Zip > Extract here.

### 3.2 Open RStudio (and configure git.exe path again)

In Windows Explorer, navigate to H:\esm296-4f\github and double-click on the github R project file to launch RStudio with that project loaded.

Unfortunately the configuration settings in Rstudio that you set in the [wk1/git](#) do not seem to transfer between machines. The only setting you really need to apply is

RStudio menu Tools > Global Options > Git/SVN > Git executable: Browse... and paste C:\Program Files (x86)\Git\bin\git.exe to look like this:

If the Git executable path wasn't already set, then you'll need to close RStudio and reopen H:\esm296-4f\github\github.Rproj.

You should only have this one instance of RStudio open for working on the labs. You'll use RStudio to write up your lab report and manage Git to commit and push to Github.

Aside: **.gitignore**

You might've noticed the **.gitignore** file which RStudio automatically creates for telling git to ignore some temporary R files. Here's the contents of the default file when you create an RStudio project in a git repository.

```
.Rproj.user  
.Rhistory  
.RData
```

You don't want to track any of these files:

- **.Rproj.user** refers to a folder RStudio creates to store internal temporary info for your session.
- **.Rhistory** file keeps your local command history from the R console.
- **.RData** stores all your environment variables for your session if enabled in RStudio options.

It's a good idea to add this **.gitignore** file, commit and push to github so that this setting takes effect next time you might clone the repo to a different machine / path.

### 3.3 Inspect and Run Lab 1 Model

A single model has been created for you which replicates lab 1. Let's have a look by opening ArcMap and saving a new blank ArcMap document to H:\esm296-4f\labs\lab2\lab2.mxd. Once you save the document the other contents in the containing "Home" folder become readily visible in Catalog. Now from within ArcMap's Catalog pane you should see lab2.tbx, right-click the **wind\_model\_Params** model > Edit (which is generally how you open models to work on).

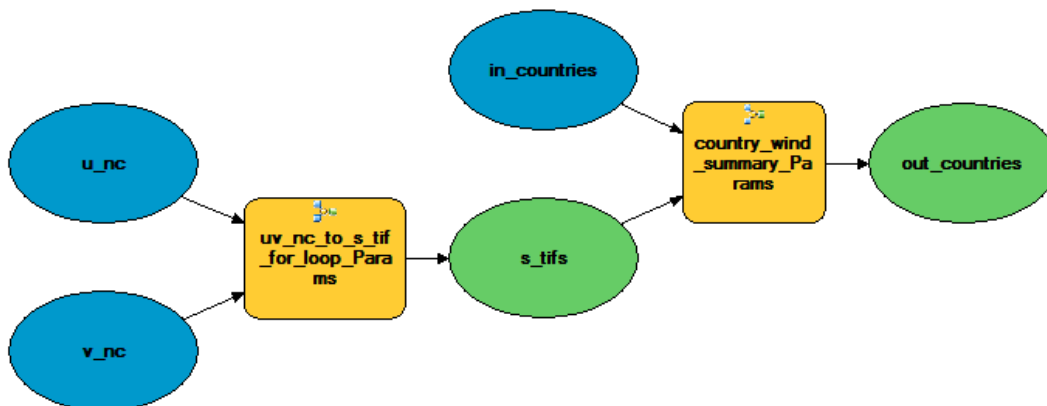


Figure 4:

### 3.3.1 Inspect Sub-Models

Notice how the “tools” (yellow boxes) are actually models (🔧). You can right click > Edit these tools to open the “sub-models”. We had to separate the models because in the case of an iterator, all tools in the model are run at every iteration. Since we want to run the summary tools just once, this model had to be separated. The multiple outputs from the first model are gathered with the [Collect Values](#). Here’s an [example](#) of iterating over rasters in a folder with a sub-model.

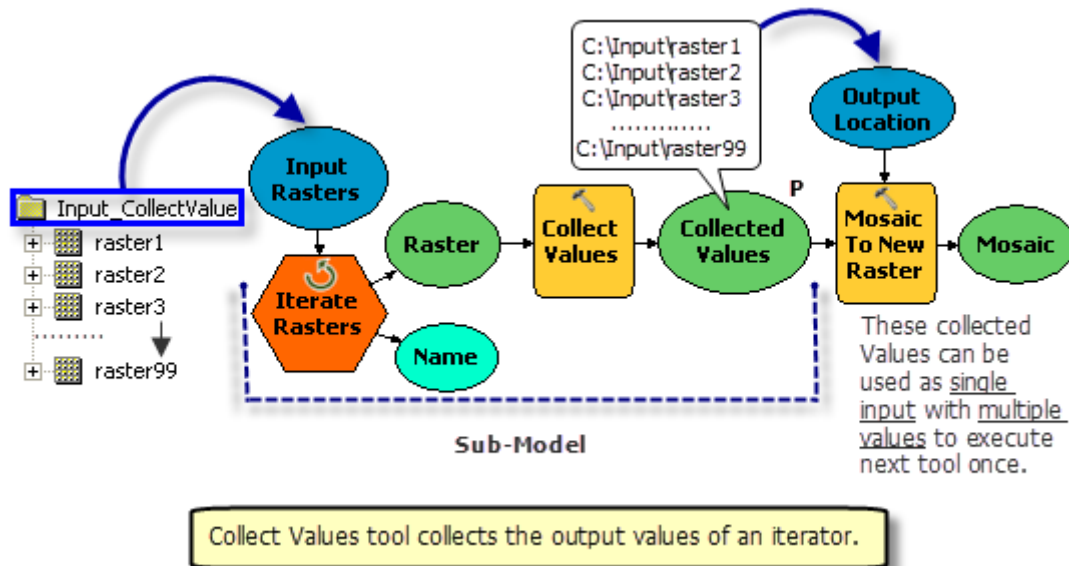


Figure 5:

Notice also the use of [model parameters](#) denoted with the big **P** in the sub-models and how these get applied by the main model. You can double click a sub-model to see its parameters.

For more details on how to do this yourself, check out the [Tutorial: Creating tools with ModelBuilder](#). We want to move onto programming and not dwell further in Model Builder world, so merely introducing you to this functionality by example for now.

### 3.3.2 Slight Differences in Model

Besides this nesting of sub-models, we’ve made a few minor modifications:

- Copied the raw/\*countries.shp into the geodatabase with the Copy Features tool. This is preferred in principal so that your analysis is completely reproducible and therefore shouldn’t modify the original raw data. Some of you may have also noticed that every time you ran the model it would join a new set of fields, not replace any existing, so the countries shapefile would end up with multiple fields (ie MEAN, MEAN\_1, MEAN\_12, etc). A fresh copy of countries now gets copied over to the geodatabase at the beginning of the model which avoids this problem.
- For loop starts j at 0. This is because looking again at the documentation for the [Make NetCDF Raster Layer](#) you’ll notice:

BY\_INDEX — The input value is matched with the position or index of a dimension value. The index is 0 based, that is, the position starts at 0.

### 3.4 Export Models to Scripts

The model above gets the job done, but it is a bit convoluted with nested models and the inline variable substitution. Really, this analysis motivates writing a script since that environment is much more suited to iterating through a loop and using variables.

So you can easily export a ModelBuilder model to a python script, however models with Iterators do not work and nested models don't reveal the submodels they use. So to get started, you'll want to export from the lab2.tbx model to a Python script (Model Builder menu Model > Export > To Python Script...) the following (need to first create folder 'H:\esm296-4f\github\lab2'):

- uv-nc\_to\_s-tif to H:\esm296-4f\github\lab2\uv-nc\_to\_s-tif.py
- country-wind\_summary to H:\esm296-4f\github\lab2\country-wind\_summary.py

We'll implement looping in the first and append to the second for a script that does it all.

### 3.5 Open Script in WinPython

Take a look at the file H:\esm296-4f\github\lab2\uv-nc\_to\_s-tif.py by right clicking on it in Windows Explorer > Edit with Pythonwin.

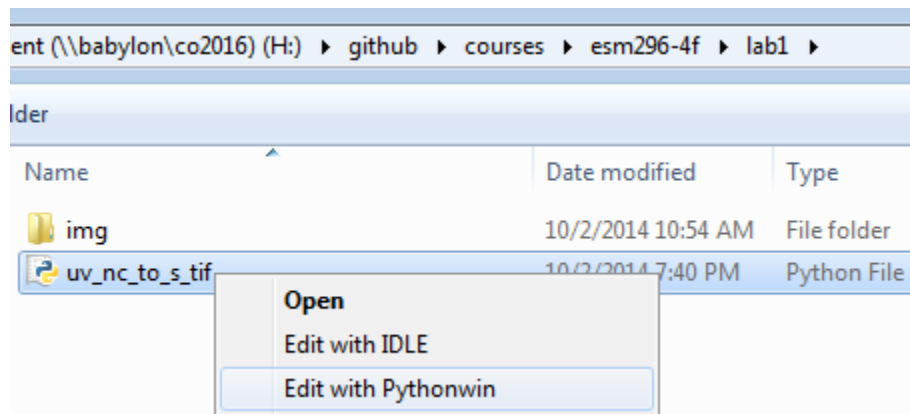


Figure 6:

A few observations on Pythonwin:

- Two windows open:
  1. Script editor to the file you opened, and
  2. Interactive Window which is the console into which you can type Python commands. Go ahead with your first test of Python after the prompt ">>>":

```
1 + 1
```

Yay, it can at least work as a simple calculator.

- Use menu Window > Tile to fill out the window with all open windows, and upper right window buttons to minimize / maximize / close.

## 3.6 Anatomy of an Arcpy Script

Once again, the “#” character begins a comment. Note the commented names for the major sections of the Python script:

- **# Import arcpy module**

First, the arcpy module gets imported. This module contains all the tools available in ArcGIS. Notice how it prefixes all the remaining statements besides the Local variables section. This `module.function()` notation is common amongst object oriented languages. More generally, you’ll see `object.function()` or `object.property`. Functions generally have arguments that are specified within the paranthesis and seperated by commas (except a small handful of special ones like `import`).

- **# Check out any necessary licenses**

In order for tools from the Spatial Analyst toolbox to run the extension needs to be checked out (equivalent to needing to tick the Spatial Analyst extension on in the ArcMap menu Customize > Extensions...).

- **# Local variables**

Notice how variable is on the left, equal sign “=” means assignment, and value on the right. In this case all variables are strings, hence wrapped in quotes. If they were strings and didn’t have quotes the values would have to either be numeric or the name of another assigned variable.

- **# Process: \***

Here the tools are doing the actual work. Notice the correspondence between the individual arguments set here and how you filled out the parameters of the Model Builder forms. Strings are used for all arguments except the input and output paths, which are assigned to variables. In fact any argument can be assigned a variable.

So what’s the advantage of assigning a variable? It makes it easy to change in one location and use in multiple locations. If you don’t expect it to change and are only using in one place then you probably don’t need to assign it to a variable; if it’s not going to “vary” then no need for a variable.

For every tool in ArcGIS there is a corresponding set of documentation and examples for its use in Python at the bottom of every tool’s help documentation. For example, check out the documentation on the first tool [Make NetCDF Raster Layer \(Multidimension\)](#). The functions generally have the form CamelCase-ToolName\_toolboxshortname, hence “MakeNetCDFRasterLayer\_md” which is part of the Multidimension toolbox (“\_md”).

## 3.7 Intro to Python

Let’s start by typing some code (not copy / paste) in the Pythonwin’s Interactive Window.

Assignment of variables is with = whereas testing equality is == which returns True or False. Try the following.

```
1 + 1
a = 1
b = 2
a == 1
a == b
c = 'a string'
a + c
```

That last line should give you an error `TypeError: unsupported operand type(s) for +: 'int' and 'str'` since you cannot add an integer and a string. You can however convert a number to a string and add two strings together. Note that quotes can be either " or ' which are functionally equivalent.

### 3.7.1 String Formatting and Variable Substitution

Next, try:

```
str(a) + c
str(a) + ' ' + c
'%s %d' % (a, c)
'%s %03d' % (a, c)
```

The second two lines use [string formatting operations](#) in which the % character is special within a string, %s for string or %d for digit, and outside it separates the corresponding arguments. In the last case %03d, you can zero-pad digits, which is useful for enabling files to nicely sort (like julian days of our example).

Let's try a couple more examples which you'll need to fully understand to translate your exported Python code into working code that updates values based on each iteration of a loop.

```
j = 1
"slice %d" % j
"slice %d"
```

Notice how in the first instance %d is replaced by the value of j. But in the second, without a variable j and extra % outside the string "slice %d" to indicate substitution the string "slice %d" is taken literally. So the presence of a % after the string "slice %d" sets up the string to accept input variables for substitution, ie "%d" to swap out for the value of variable j, which gets formatted as "1".

### 3.7.2 Module Import, Whitespace, For Loop, Zero-Based Indexing

Let's try importing a module.

```
import antigravity
```

Ok, that's a little Python joke (should've launched a web browser to [xkcd.com/353](http://xkcd.com/353)). It does point to a couple interesting aspects of Python though.

1. Dynamic Typing. This just means that Python figures out how to store the variable in memory without needing to be overly verbose as to what type of variable it is and allowing it to safely interact with other variables. (More details [here](#).)
2. Whitespace. In Python whitespace matters. Indentation means that the code executes within the statement (for loop, conditional if, etc) as what precedes it.

To see how whitespace matters, try typing this.

```
for i in range(1, 13):
    print i
```

Be sure to hit an extra return at the end. Congratulations, you just executed a loop. Notice how the interpreter automatically indented for you and the print statement executed for every iteration of the loop. Also notice how i only goes up to 12, not 13. To see why, look at the help for this function.



```
help(range)
range(13)
```


Python is zero index based, ie the first element is indexed by 0 (not 1). So it starts at 0 and ends one less than the end. To extract a given element from a list, you would use an index starting with 0.

```
a = range(13)
a[0]
a[1]
a[13]
```

That last indexing should bonk with `IndexError: list index out of range` since index 13 would actually reference the 14th element of the list which is not generated by `range(13)`. It's confusing at first, but makes sense the more you use. (In contrast, the indexing for R starts at 1, not 0.)

### 3.8 Python Window in ArcMap

While Pythonwin is a decent editor, you can't simply copy/paste more than one line of code from the Editor window to the Interactive Window (although you can copy multiple lines, right-click in Interactive Window and "Execute python code from clipboard").

I tend to edit code in Winpython and run chunks of code in ArcMap's [Python window](#) (available from the main toolbar as ) which does allow copy and paste of multiple lines of code and shows you the results in ArcMap.

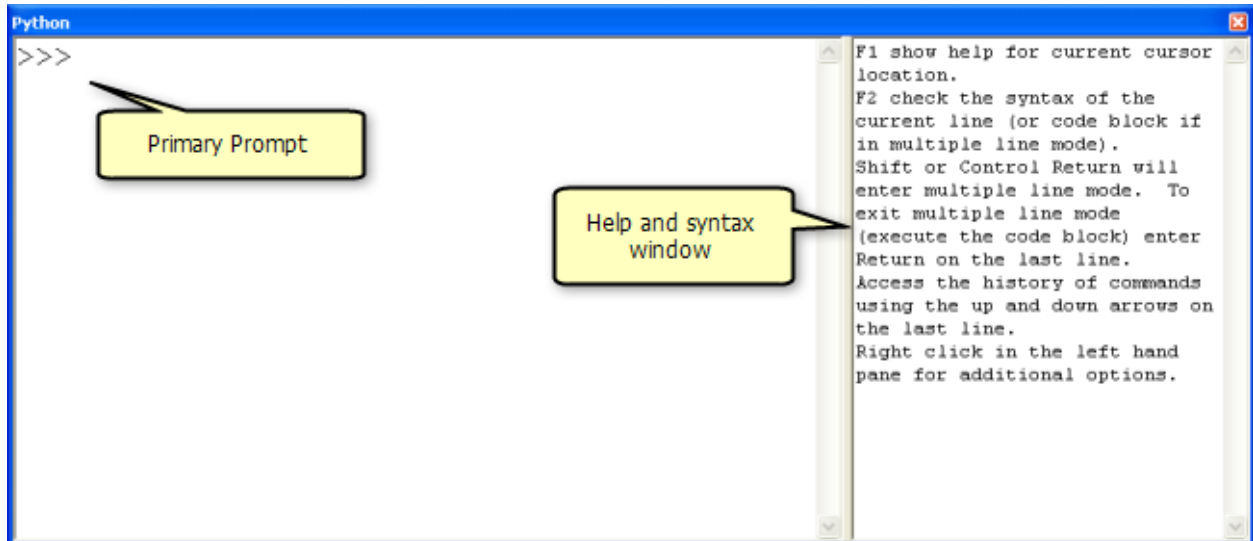


Figure 7:

Launch the Python Window in ArcMap and try out the following mostly explained in [Using the Python window](#).

```
# hello world
print "Hello Jack!"
count = 0
count
```

```
# assignment
x = 1
y = 2
print x + y

# a loop. note the indentation
for count in range(1,5):
    print count
```

Paths are usually defined as strings. As already mentioned in Microsoft’s “infinite wisdom” decided to use a new file separator, and instead of forward slash / (like everyone else) they went with the backslash \, reserved as the special escape character in most all other languages (eg try `print 'testing \n\t some \tbackslashing'` to see newline and tab characters printed). This turns out to be a regular annoyance. Here are 3 ways to set the same file path to variable `shp`.

```
shp = "H:\\esm296-4f\\labs\\lab2\\raw\\World_EEZ_v8_2014.shp"
shp = "H:/esm296-4f/labs/lab2/raw/World_EEZ_v8_2014.shp"
shp = r'H:\esm296-4f\labs\lab2\raw\World_EEZ_v8_2014.shp'
```

These all work the same when passed to a function. The last one uses the prefix `r` for raw format so doesn’t interpret the escape character in a string, which makes it rather convenient to easily copy and paste from Windows Explorer or Arc Catalog address bar. Could’ve interchangeably used single quote `'` or double quote `"` - I like the single quote since it’s less bulky.

Assuming you set the variable `shp`, try importing the `arcpy` module and get a count on how many rows in this feature class.

```
import arcpy
arcpy.GetCount_management(shp)
```

You can type a function in the ArcMap Python Window (where it will try to auto-complete for you) and on the right hand side it will provide the help documentation for that function.

**Tip: Accessing Python Snippets in Geoprocessing Results Pane**

I also find it handy to run any tool and then go into the [Geoprocessing Results](#) pane (menu Geoprocessing > Results), right click on an executed tool and Copy As Python Snippet to get the function name and all its parameters.

Use this trick to add the final three lines to your script. Be sure to navigate to the filesystem path of all the inputs and not use layers only temporarily available in ArcMap memory. This will make the Python snippet reusable in the future and prevent comments like this in your snippets:

### 3.9 Add Loop in Pythonwin

Let’s now insert a loop in the first model builder dumped script. To do so, return to Pythonwin’s editor window with the file `H:\esm296-4f\github\lab2\uv-nc_to_s-tif.py` and save it (File > Save As...) to `H:\esm296-4f\github\lab2\wind_script.py`.

Let’s add a loop with the same parameters as the For loop in our Model Builder model (from 0, to 365, increment 30). Every variable with a “001” suffix should use string formatting to substitute values, so needs to be moved down below the for statement. And everything to be looped needs to be indented.

Here’s part of the answer, which you can copy and paste into ArcMap’s command window to try out. (Should print out 13 lines with the unique filename updated having an incrementing zero padded value of `j` substituted.)

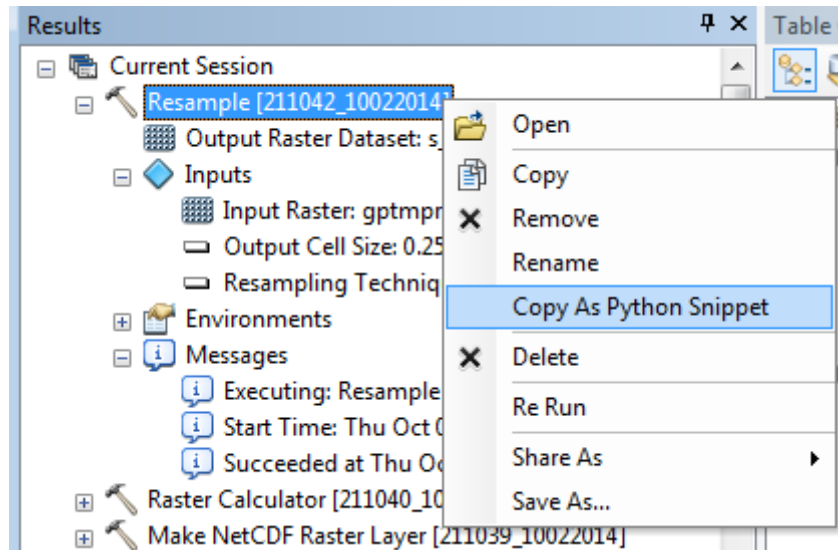


Figure 8:

```
# loop over a range of values for j
for j in range(0, 365, 30):

    # assign values based on variable j
    s_001_tif = "h:\\esm296-4f\\labs\\lab2\\out\\s_%03d.tif" % j
    print s_001_tif
```

You'll want to place this loop into the script after defining `u` and `v`, and before the process tools.

Go ahead with moving and updating with the same string formatting trick the rest of the variables having names ending in “\_001” (`u_001`, `v_001`, `s_001`, `s_001_tif`).

It's very important you also use variable substitution for the index, currently “time 1”, so each iteration extracts a new slice of data from the netcdf. Otherwise every iteration will simply be the first month's data. You don't want zero padding in this case since it's not formatting a filename to be neatly sorted, but used as an input parameter to the tool.

You'll need to also update the formula in the Raster Calculator step from:

```
"SquareRoot( Square(\"%u_001%\") + Square(\"%v_001%\") )"
```

to:

```
"SquareRoot( Square('%s') + Square('%s') )" % (u_001, v_001)
```

The Model Builder syntax for inline variable substitution (wrapping the variable in percent symbols like `%u_001%`) no longer works in the Python environment, so you're using the same Python string formatting approach to update the formula with values of the `u_001` and `v_001`, which are getting updated at every iteration `j` of the for loop. Note that originally to include the quote inside a quoted string, the escape character was needed. Another trick is alternating the quote character (from “” to ‘’, or vice versa) so escaping is not needed.

Since you might need to run this script more than once to debug any syntax issues or missing iterators, I recommend adding the following line to set your `arcpy` environment to allow overwriting output.

```
arcpy.env.overwriteOutput = True
```

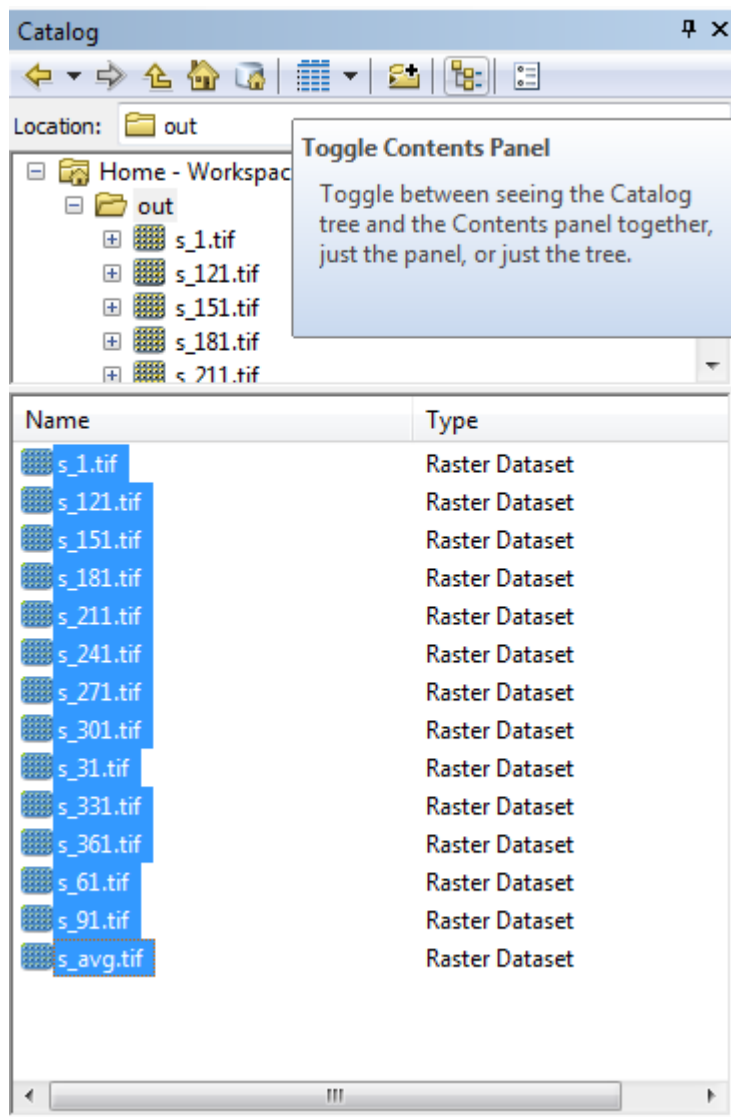
A good place for this is below the `arcpy.CheckOutExtension`, after import `arcpy` and before executing any tools where other `arcpy` environments are getting set.


Ok, almost ready to run. First:

- **Delete contents** from the `H:\esm296-4f\labs\lab2\out` folder so you can be assured it's generating the output.

#### Tip: Selecting Multiple Items in Catalog


You can toggle the contents panel in the Catalog window of ArcMap to enable multiple select (left-click top, shift and left-click bottom of list), which is very handy for deleting or moving many items at once.



- **Check syntax** (File > Check or  button). If it completes successfully, you'll see at the status bar at the very bottom of the application window "Python and TabNanny successfully checked the file"

and if not then “Failed to check - syntax error” and will move your cursor to where the first problem encountered.

- **Close Arcmap** so you don’t get file locks.

If the syntax checks OK, then you can **run** the whole script from Pythonwin (File > Run... or  button). You’ll get a prompt like this, for which you can simply accept the defaults for now and OK.

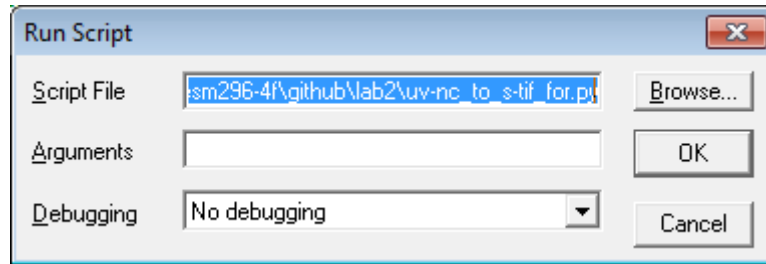


Figure 9:

If it works correctly, you’ll get output to the Interactive Window like so:

```
h:\esm296-4f\labs\lab2\out\s_000.tif
<string>:1: SyntaxWarning: import * only allowed at module level
h:\esm296-4f\labs\lab2\out\s_030.tif
<string>:1: SyntaxWarning: import * only allowed at module level
```

That extra prompt about the SyntaxWarning is related to how Model Builder exports an old tool for performing the Raster Calculator. You’ll be shown later how arcpy’s Spatial Analyst module now expects to do handle [Building complex statements](#) using map algebra.

#### Issues?: Use Github To Communicate

Github is really built to communicate about code. If you have trouble running your script and we’re outside lab or office hours when you can easily enlist our over the shoulder help, use Github to commit and push your code onto your repository. Then you can add a comment to the lab 2 issue, like this:

Hi @bbest,

Can you help me understand these lines of code:

[https://github.com/bbest/esm296-4f/blob/master/lab2/uv-nc\\_to\\_s-tif.py#L17-L18](https://github.com/bbest/esm296-4f/blob/master/lab2/uv-nc_to_s-tif.py#L17-L18)

The syntax check errors out and puts my cursor there, so something must not be right and I suspect the c

You won’t be able to see this URL because it’s my private repo, but I pulled it by visiting one of the Python scripts in a web browser on my Github repo and selecting line(s) with left click (shift and click again to select multiple lines). Notice the lines ‘#L17-L18’ added to the end of the URL.

You can also visit any Commit (left top nav menu below repository description) and comment on lines added/deleted/modified like so:

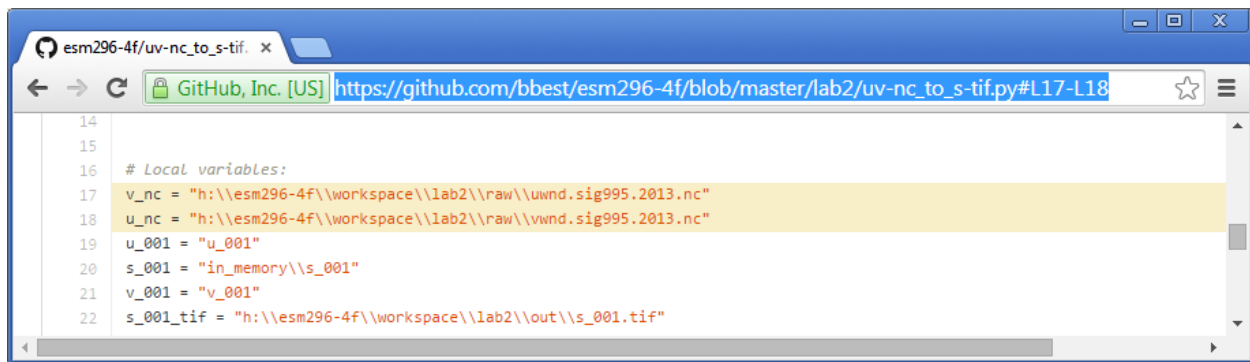
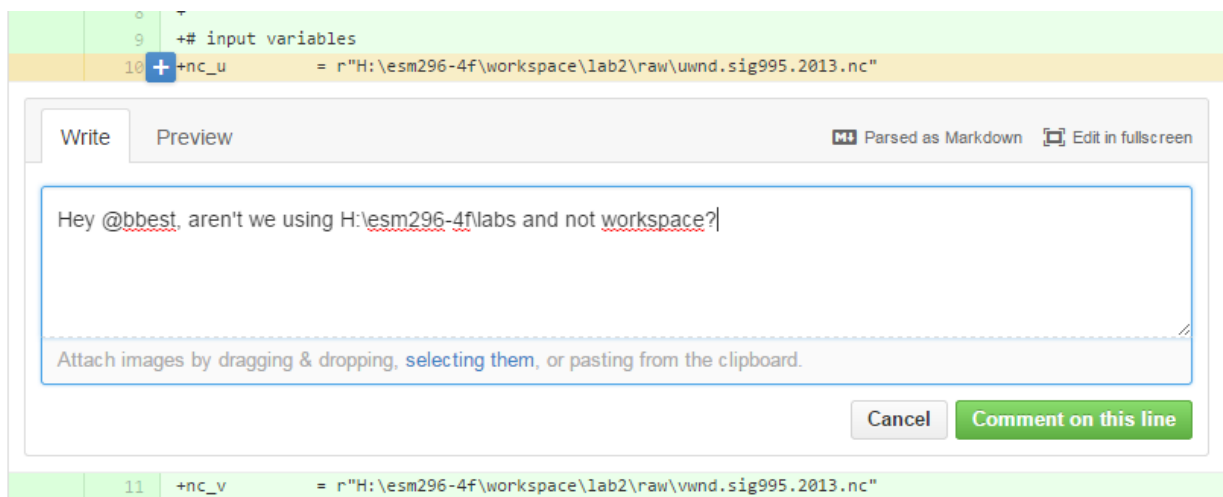


Figure 10:



Tip: **Working with Many Layers in ArcMap** If you have many layers in an ArcMap document, it can easily get overwhelming. Here are a few tricks to manage that.

- **Making just one layer visible.** You can quickly turn just one layer on by holding Alt + left click on the layer of interest. This turns off (ie unticks) the other layers.
- **Collapse All Layers.** You can also right-click the root top of the contents Layers > Collapse All Layers to reduce them to their smallest viewable name.
- **Pause Drawing.** At the bottom of the map pane is a pause button, which frees up ArcMap for other purposes.

### 3.10 Add Summary Functions to Script

Next, you'll add the summary functions in H:\\esm296-4f\\github\\lab2\\country-wind\_summary.py to the H:\\esm296-4f\\github\\lab2\\wind\_script.py script. The first two lines of this country-wind\_summary (import, arcpy.CheckOutExtension) are redundant and shouldn't be added again. You'll want to add the Local variables above and outside the for loop, and the process scripts to the bottom, dedented (ie aligned flush left), after and outside the for loop.

Here's a nice Pythonic way (ie a [list comprehension](#)) to set a variable to being the list of all the s\_#\_tif rasters. So you'll want to replace these two chunks of code.

from:

```

s_0_tif = "H:\\esm296-4f\\labs\\lab2\\out\\s_0.tif"
s_120_tif = "H:\\esm296-4f\\labs\\lab2\\out\\s_120.tif"
s_150_tif = "H:\\esm296-4f\\labs\\lab2\\out\\s_150.tif"
s_180_tif = "H:\\esm296-4f\\labs\\lab2\\out\\s_180.tif"
s_210_tif = "H:\\esm296-4f\\labs\\lab2\\out\\s_210.tif"
s_240_tif = "H:\\esm296-4f\\labs\\lab2\\out\\s_240.tif"
s_270_tif = "H:\\esm296-4f\\labs\\lab2\\out\\s_270.tif"
s_30_tif = "H:\\esm296-4f\\labs\\lab2\\out\\s_30.tif"
s_300_tif = "H:\\esm296-4f\\labs\\lab2\\out\\s_300.tif"
s_330_tif = "H:\\esm296-4f\\labs\\lab2\\out\\s_330.tif"
s_360_tif = "H:\\esm296-4f\\labs\\lab2\\out\\s_360.tif"
s_60_tif = "H:\\esm296-4f\\labs\\lab2\\out\\s_60.tif"
s_90_tif = "H:\\esm296-4f\\labs\\lab2\\out\\s_90.tif"

arcpy.gp.CellStatistics_sa("H:\\esm296-4f\\labs\\lab2\\out\\s_0.tif;H:\\esm296-4f\\labs\\lab2\\out\\s_1

```

to:

```

s_tifs = ["h:\\esm296-4f\\labs\\lab2\\out\\s_%03d.tif" % j for j in range(1, 365, 30)]

arcpy.gp.CellStatistics_sa(s_tifs, s_avg_tif, "MEAN", "DATA")

```

Try running the first line above in the Interactive Window. Then run `s_tifs` to show the result. Next: `type(s_tifs)`. Let's inspect the length of the list with `len(s_tifs)`.

Notice that we're now feeding to the first argument of the [Cell Statistics](#) function a list rather than a string. This is a different format than the original single string with paths delimited internally by semicolons. ArcGIS functions that allow multiple inputs for a given argument will generally accept either, but it's nicer programmatically to be able to manipulate a list already separated out by items. So with a list comprehension, you're essentially running a for loop to generate the list. Here's an equivalent (and more verbose) way to create this list.

```

s_tifs = []
for j in range(0, 365, 30):
    s_tifs.append("h:\\esm296-4f\\labs\\lab2\\out\\s_%03d.tif" % j)

```

Note that we have to “instantiate” the list (ie create an instance of it, in this case empty) so that we can then append to it. All lists have the append method. You can inspect which methods and properties are available for any given object with the `dir()` function in Python `dir([])` and then for more on that function `help([].append)`.

Ok, repeat the Delete contents, Check syntax, Close ArcMap steps as above before running the whole script again.

### 3.11 Change to EEZ data

So far the script is calculating the terrestrial wind. Now update the `in_countries` variable in your script to use the marine exclusive economic zone (EEZ) described above in the Data section. You'll need to also update the field parameter used to define your zones and for the join (ie the field “NAME” does not exist in the EEZ shapefile; you can look at its table from ArcMap to determine a reasonable replacement field name).

Repeat with Delete contents, Check syntax, Close ArcMap steps as above before running the whole script again.

### 3.12 Turn in Report

Based on the output countries from your script, produce a similar [map](#) and markdown [report](#) as lab1 but now for offshore wind and at H:\esm296-4f\github\lab2\README.md with:

- map of EEZ colored by max (no need to do the model builder images)
- top 3 countries by max and mean
- link to your final wind\_script.py script

When you [commit](#) this README use the message to close your lab 2 issue (or visit the issue and close it separately).

## 4 Review

You should now have familiarity with these techniques.

- **ArcGIS Advanced Model Builder**
  - **Models as Tools** from Model Builder enabled the nesting of one parameterized model with the other.
- **ArcGIS Python**
  - **Export model as Python script**
  - **Anatomy of an arcpy script**
  - **Python window** for execution of Python code
  - **Geoprocessing results** pane and Python snippets
  - **Arcpy** module, environment (**overwriteoutput**) and toolboxes ()
- **Python programming**
  - **variable** assignment =, comparison ==, and data types **str**, **int**, **list**
  - **module import**
  - **whitespace** matters for iteration
  - **zero indexing**
  - **iteration for**
  - **string formatting** and variable substitution 'a %s string and %d digit' % ('abc', 3)
  - **list comprehensions** ['example %d' % i for i in range(1, 10)]
  - **PythonWin** as an editor, syntax checker and run environment (aka integrated development environment (IDE))