Computer Lab 3a: Lists, arrays

Climate Data Analysis, ATS 301, Fall 2018

Primary objectives for this lab:

- Learn about lists, numpy arrays

First, though, a reminder about your new favorite command:

Help function

The `help` command is one you will probably use quite often.

In [ ]:
```
help(print)
```

The `help` command provides information about input values, options, etc. After you import a module, you can use `help` to learn about specific commands:

In [ ]:
```
%matplotlib inline
import matplotlib.pyplot as plt
help(plt.plot)
```

Commenting

Sometimes you want to include information within a chunk of code, without evaluating that text. I've already used *commenting* in some cells of Lab 2.

- Comments in Python are indicated by `#`
- Any line that starts with `#` will be ignored.
- If `#` appears later in a line, the part after `#` will be ignored.

In [ ]:
```
# This is a comment. It is not evaluated.
```

In [ ]:
```
print("Hello, World") # The first part of this line is evaluated, but not the part after #
```

In [ ]:
```
print("This")
# print("That")
```

Even though the second print command is a valid command, it is not evaluated.

Types of Variables

Values and variable have different types

- Integer (2, 17, 555, etc.)
  - class 'int'
- Floating-point number (1.5, 0.776, etc.)
  - class 'float'
- String ('Hello, World!')
  - class 'str'

Use the `type` function to determine the type if you're not sure:

In [ ]:
```
type(2)
```

In [ ]:
```
type(2.0)
```

Lists

Basic Python includes another built-in type: lists

A *list* is a *sequence* of *values*. Here are a few examples of lists:

```
[1, 5, 7, 10]
["a","b","c","d","e"]
[1, "yellow", 5, "f"]
```

Note that, technically, you can have a mix of different *types* within a single list. In general, I don't recommend doing this. In fact, we're pretty much going to use arrays for everything, which we'll get to in a little while.

- The square brackets ("`[`" and "`]`") indicate a list.
- *Values* within the *list* are separated by commas.
- You can assign lists to variables.

A few things you can do:

Create a list:

In [ ]:
```
my_list = [7, "blue", 42.5]
my_other_list = ["a", 3.14, "Karen"]
```

Combine two lists:

In [ ]:
```
my_big_list = my_list+my_other_list
print(my_big_list)
```

To **create an empty list** using empty square brackets (useful if you want to initialize a variable before adding values to it):

In [ ]:
```
my_empty_list = []
```

You can then **add (append) things to a list** by using the `append` function:

In [ ]:
```
my_empty_list.append(12)
print(my_empty_list)
# not so empty anymore
```

Appending can be useful when you are repeating a calculation and want to save the result each time, for later analysis or plotting.

List Indices

To use a specific value from a list, specify the list *index*:

In [ ]:
```
print(my_big_list[2])
```

Does this give you the value you expect?

List indices start at 0, not 1. So the first value in a list is `my_big_list[0]`.

You can **replace individual values** within a list:

In [ ]:
```
my_big_list[2] = 12
print(my_big_list)
```

Important Note:

- **Functions use `()`**
- **Lists/arrays use `[]`**

But really, lists are not very useful for our purposes. Instead, we want ...

Arrays

Arrays are widely used in scientific computing to store the values of a variable with respect to time, space, or some set of parameters. The default functions available in Python are not very useful, so we use the package `numpy` to work with arrays.

Arrays:

- Way to store data in orderly fashion
- Similar to lists, except that every element of an array must be of the same type, typically a numeric type like `float` or `int`
- Like matrix
- Can be any number of dimensions (lists are only one)

For more info on arrays:

- One good resource is "A Hands-On Introduction to Using Python in the Atmospheric and Oceanic Sciences" http://www.johnny-lin.com/pyintro/ed01/free_pdfs/ch04.pdf (http://www.johnny-lin.com/pyintro/ed01/free_pdfs/ch04.pdf) (which is also available in the "Resources" directory on the server, `ch04.pdf`).
- Also `numpy.pdf` (Introduction to Numpy and Scipy, also on jupyter)
- Under the Help menu from a Jupyter notebook, you'll find a link to the numpy web page (as well as some other reference pages).

To use `numpy`, we have to first import the module. To make the name we use in our code a little shorter, a common option is to import the module as "`np`". You'll see this in lots of examples online.

In [ ]:
```
import numpy as np
```

If you already have a list, you can **convert it to an array** using np.array. Here's an example from this web page: http://www.python-course.eu/numpy.php/ (http://www.python-course.eu/numpy.php/)

Start with a regular *list* with values of temperatures in Celsius:

In [ ]:
```
cvalues = [25.3, 24.8, 26.9, 23.9]
```

Use the `np.array` function to turn this list into a one-dimensional `numpy` array:

In [ ]:
```
C = np.array(cvalues)
print(C)
```

Now, convert the values into degrees Fahrenheit. Try the *array* first:

In [ ]:
```
print(C * 9 / 5 + 32)
```

What happens when you try the same calculation with the *list*?

In [ ]:
```
print(cvalues*  9 / 5 + 32)
```

So, using `numpy` makes working with 1-D arrays much easier (as well as allowing multi-D arrays.]

You can create an array in different ways. As shown above, you can create an array by converting a list. You can also create a multi-D array by using a *nested lists* (a list of lists).

In [ ]:
```
A = np.array([
[11,12,13,14,15],
[21,22,23,24,25],
[31,32,33,34,35],
[41,42,43,44,45],
[51,52,53,54,55]])
```

In [ ]:
```
print(A)
print(np.ndim(A))
print(np.shape(A))
```

`np.ndim` returns the number of *dimensions* in the array, while `np.shape` returns the sizes of each of those dimensions.

What happens if you leave off the `np.` ?

In [ ]:
```
print(ndim(A))
```

You can also create arrays with the `np.ones` function, which takes the *shape* of the array as input and fills the array accordingly with ones.

In [ ]:
```
E = np.ones((2,3))
print(E)
```

There is also an `np.zeros` function, while does what you'd expect:

In [ ]:
```
Z = np.zeros((2,4))
print(Z)
```

Array Indexing

To get a particular value in the array, specify the index for each dimension. **Remember that indices start at 0, not 1.**

For 2-D arrays, the array indices are separating by a comma: `[row index, column index]` . For example, this prints the value in the second row of the first column:

In [ ]:
```
print(A[1, 0])
```

Which set of indices gives the value of 34?

In [ ]:
```
# Replace the ???? with the correct indices here.
print(A[????,?????])
```

Array slicing

To get all values in a given row or column, use *slicing*.

In [ ]:
```
print(A[1,:])
```

The `:` in the second index position indicates inclusion of all of the values in that row (dimension).

How would you specify all the values in the first column?

In [ ]:
```
print(A[????,?????])
```

You can also specify a *subset* of a *slice*.

The general syntax for subsetting a dimension is

```
[start:stop:step]
```

- `start` is the first index that will be used (inclusive)
  - If you do not specify the start index, it will be 0.
- `stop` is when to stop the subsetting. It is **exclusive** (i.e., the value for that index is not included).
  - If you do not specify the stop index, the subset will include the last value (`stop` will be the last index + 1).
- `:step` is optional. If you do not select a step, it will be 1.
- An **negative index** indicates counting from the end of the row/column (e.g., "-1" means the last index.) This is helpful when you don't remember how big the array is and want to, for example, take the last *n* values.

Before you try each of the following, what you you expect the result to be?

(Note that you can leave off `print()` if you evaluate each cell {shift-ret} separately.)

In [ ]:
```
print(A[2:5,:])
```

In [ ]:
```
print(A[:4,:])
```

In [ ]:
```
A[4:,:]
```

In [ ]:
```
A[5:,:]
```

In [ ]:
```
A[:,:]
```

In [ ]:
```
A[0:5:2,:]
```

In [ ]:
```
A[3:-1,:]
```

You can use subsetting for both rows and columns at the same time.

How to specify 2nd and 3rd values in row 3?

First three values in rows 4 and 5?

In [ ]:
```
# Your code for these two cases here.
```

In [ ]:
```

```