

Python Numpy Intro

This tutorial is an introduction to the Python Numpy numerical python library.

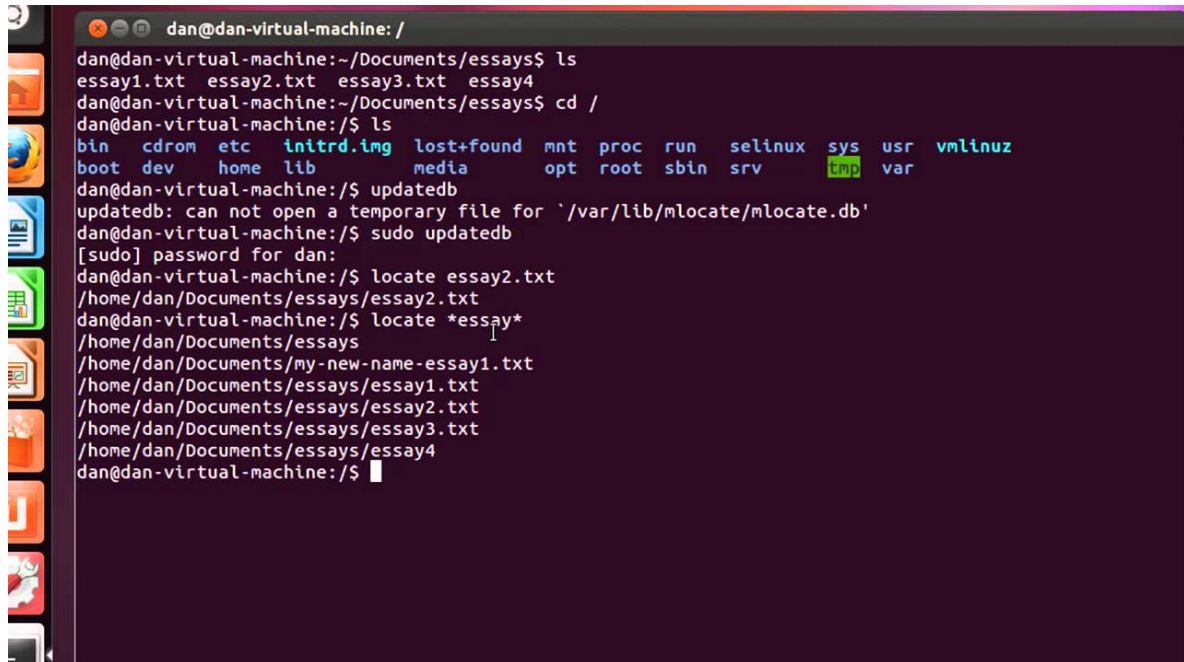
Python comes with lots of features built-in to it's native language. However, people will also often use what's called packages. These are external pieces of code, written by other people, which we can download and use for free (they are sometimes also referred to as libraries). There are literally thousands of packages out there, but typically data scientists will only be using 5 or so regularly. Today we will be looking at NumPy (short for Numerical Python, pronounced *num-pie*).

The core data structure behind Numpy is the n-dimensional Numpy Array. It is 3x to 10x faster and more memory efficient than Python's lists we have looked at before because, similar to Java arrays, it uses contiguous blocks of memory, and all elements are the same data type so there is no type checking at runtime. The Numpy library also includes many built-in code-saving mathematical functions that can be performed on an entire array or any slice of an array with a single line of code (ie. no for loops).

Numpy n-dimensional arrays are also sometimes referred to as nd-arrays.

Installing NumPy

When you download Python for the first time, none of these packages will be installed. Packages have to be installed from the *command line*. You may have seen this briefly already. It looks something like this.

A screenshot of a terminal window titled 'dan@dan-virtual-machine: /'. The user is in the directory ~/Documents/essays and lists files: essay1.txt, essay2.txt, essay3.txt, and essay4. They then run 'cd /' and 'ls', showing the root directory contents including bin, cdrom, etc, initrd.img, lost+found, mnt, proc, run, selinux, sys, usr, vmlinuz, boot, dev, home, lib, media, opt, root, sbin, srv, tmp, and var. Next, they run 'updatedb', which fails with an error about a temporary file. They then run 'sudo updatedb', enter their password, and use 'locate' to find 'essay2.txt' and all files containing 'essay' in their names, showing their full paths in the /home/dan/ directory.

The command line can be reached by typing `cmd` into your windows search bar. Packages can be installed by typing the following into a command line and hitting enter.

```
pip install package_name
```

where you replace `package_name` with the actual name of the package you want to download. This should always work if you have python installed.

So, our packages can be installed by typing:

```
pip install numpy
```

The import statement

Once we have a package installed and we want to use it in our code, we have to *import* it. This is achieved as follows:

```
import package_name    # this is now regular python code!
```

Lets try that now

```
import numpy as np
# The 'as np' statement will rename numpy to np for short.
np.set_printoptions(precision=3, linewidth=500, threshold=500, suppress=True)
# dont worry about the line above, it just makes some printed stuff look
prittier.
```

If you have no errors, that means you imported it fine. We can now access NumPy functions by calling `np` followed by a full-stop (`.`) then the name of the function we want to use.

If you need more information about a function, just run a code block with the function name followed by a question mark, e.g

```
np.min?
```

Creating a Numpy Array

The key thing we use NumPy for is what's called a NumPy *array*. This is similar to a list, except we can only use numbers. We can make a NumPy array like so

```
a = np.array([1, 2, 3, 4])
a
```

```
array([1, 2, 3, 4])
```

NumPy arrays are great when we want to do things to lots of numbers at the same time. For example, if we multiply the array by 2, we get a new array where every item has been multiplied by 2.

```
2 * a
```

```
array([2, 4, 6, 8])
```

Or if we add 1, again it will do it to every item in the array.

```
a + 1
```

```
array([2, 3, 4, 5])
```

We can even square an array:

```
a ** 2    # Remember, ** means to the power of!
```

```
array([ 1,  4,  9, 16])
```

This is much more convenient than lists, where we would have to write a loop to achieve the same thing.

```
a_list = [1, 2, 3, 4]
a_list_squared = []
for i in range(4):
    a_list_squared.append(a_list[i] ** 2)
a_list_squared
```

```
[1, 4, 9, 16]
```

trying to square a list will give an error!!

```
a_list ** 2
```

```
-----
TypeError                                Traceback (most recent call last)

<ipython-input-18-288b1df3789d> in <module>()
----> 1 a_list ** 2
TypeError: unsupported operand type(s) for ** or pow(): 'list' and 'int'
```

We can also add or multiply two arrays together, as long as they are the same length. Again, this will be done to each item in turn:

```
b = np.array([3, 4, 5, 6])
c = np.array([7, 6, 5, 4])
```

```
b + c    # add two arrays
```

```
b * c    # multiply two arrays together
```

```
b - c    # subtract one array from another
```

Note that we cant do any of these things with lists:

```
a_list = [3, 4, 5, 6]
b_list = [7, 6, 5, 4]
```

```
a_list - b_list # Running this code will evoke an error: TypeError: unsupported
operand type(s) for -: 'list' and 'list'!
```

Problem 1

15 people take a maths test. They get a score out of 20 which has been put into a NumPy array. For example, a score of 15/20 should be a percentage of 75. Using one or two lines only, print out these scores as a percentage

```
# Write your solution here:
```

Indexing NumPy arrays

NumPy arrays can be indexed (i.e. getting elements or sub-arrays) in a very similar way to regular lists.

```
# index value:      0  1  2  3  4  5  6  7  8
my_array = np.array([5, 4, 2, 5, 3, 2, 2, 5, 1])
```

```
# get the item at index 0
my_array[0]
```

```
# get the item at index 5
my_array[5]
```

```
# get items from 3 (inclusive) to 7 (non-inclusive)
my_array[3:7]
```

```
# get items from the beginning to item 5 (non-inclusive)
my_array[:5]
```

```
# get items from item 5 (inclusive) to the end
my_array[5:]
```

Example

We can use a loop to move through a NumPy array in a similar way to lists, if we want. Here we'll create an array of the first 20 Fibonacci numbers:

```
fib = np.zeros(20)    # this creates an 'empty' numpy array of 25 zeros
print('the initial array: ', fib)

fib[1] = 1    # set element 1 to 1

# this is a 'for' loop. Ask the instructor if you're unsure about this
for i in range(2, 20):
    fib[i] = fib[i-1] + fib[i-2]

print('\nthe final array: ', fib)
```

NumPy Functions

NumPy has lots of useful functions which we use too. They are all designed to either make or operate on NumPy arrays. These are functions and work in the exact same way as the ones you've already seen. That means they are called on arguments using brackets to return something.

```
function(argument)
```

Mean

We can find the mean (average) of a NumPy array by using the function `np.mean(array)`

```
my_array = np.array([1, 2, 3, 4, 5, 6, 7])
```

```
np.mean(my_array)
```

Max/Min

We can find the minimum or maximum of a NumPy array by using the function `np.min(array)` or `np.max(array)`

```
np.min(my_array)
```

```
np.max(my_array)
```

Sum

We can find the sum of an array by using the function `np.sum(array)`.

```
np.sum(my_array)
```

Problem 2

For the following arrays, calculate the *range* (the difference between the maximum and minimum value).

```
a1 = np.array([13, 34, 21, 27, 16, 46, 14, 9, 6, 38, 8, 27, 37, 29, 48, 33,
27])
a2 = np.array([ 8, 12, 6, 4, 37, 34, 31, 33, 2, 12, 21, 27, 28, 46, 19, 28,
32])
a3 = np.array([ 5, 46, 22, 38, 28, 24, 44, 13, 45, 13, 3, 27, 23, 19, 40, 45,
22])
```

```
a1.max() - a1.min()
```

```
42
```

Linspace

The function `linspace` will make an array that has N evenly spaced numbers between a and b . We call it as

```
numbers = np.linspace(a, b, N)
```

```
a = 5    # start here
b = 30   # end here
N = 33   # take this many steps
numbers = np.linspace(a, b, N)
```

```
numbers
```

```
array([ 5.    ,  5.781,  6.562,  7.344,  8.125,  8.906,  9.688, 10.469, 11.25 ,
 12.031, 12.812, 13.594, 14.375, 15.156, 15.938, 16.719, 17.5  , 18.281, 19.062,
 19.844, 20.625, 21.406, 22.188, 22.969, 23.75 , 24.531, 25.312, 26.094, 26.875,
 27.656, 28.438, 29.219, 30.    ])
```