ST445 Managing and Visualizing Data

Python and NumPy Data Structures

Week 2 Lecture, MT 2020 - Chengchun Shi

Lecture Overview

1. Python file operations
2. Application Dependencies
3. Python Data Types
4. The NumPy ndarray
5. datetime module

Python File Operations

Basic Python offers methods of reading (and writing!) files which are simple and effective. More flexible methods are offered within NumPy and Pandas but here is how to read a text file with Python:

In [1]:

```
myfile=open('Harry Potter and the Sorcerer.txt','r', encoding="ISO-8859-1")
firstline=myfile.readline()
secondline=myfile.readline()
thirdline=myfile.readline()
print(firstline)
print(secondline)
print(thirdline)
# print(myfile.read())
myfile.close()
```

```
Harry Potter and the Sorcerer's Stone



CHAPTER ONE
```

- The second argument *r* indicates that the file is opened for *read* access. *w* indicated write access, *a* for appending.
- Use *r+*, *w+* or *a+* for both reading and writing.
- *w+* and *a+* will create a file if it does not exist.

It's good practice to close a file, but it is not strictly necessary. Not closing a file might

- slow down your program
- many changes do not go into effect until the file is being closed
- run in to limits of how many files have been opened (theoretically)

In [2]:
```
myfile=open('simultion.py', 'w')
myfile.write('print(\'hello world\')')
myfile.write('\n')
myfile.write('temperatue = 98.6')
myfile.close()
```

- Python
- SQLite

  - Numpy
  - urllib
  - Matplotlib

    - Pandas
    - scikit-learn
    - Seaborn
    - NetworkX

Dependencies

- Dependencies are the modules that you need to create and run your code
- This can include modules written by yourself
- Installed separately from system-level packages such as *math*

To install a module with Anaconda you need to use *conda* commands. For example:

> conda install Quandl

Sometimes a package is not available as a *conda* pacakge but can be installed by *PyPI*

> pip install Quandl

To find out more about conda please consult [https://docs.conda.io/projects/conda/en/latest/user-guide/getting-started.html (https://docs.conda.io/projects/conda/en/latest/user-guide/getting-started.html)](https://docs.conda.io/projects/conda/en/latest/user-guide/getting-started.html)

There's a handy cheat-sheet if you want to learn, for example, about updating packages.

Using Dependencies

The basic syntax is, *import module*

For example, using the alias np for numpy:

In [3]:

```
import numpy as np
print(np.pi)
```

```
3.141592653589793
```

System Modules

Four system (built into Python) modules that are good to know about:

1. math
2. random
3. os
4. sys

Doing the maths

In Python even to do fairly simple maths equires you to *import math*

In [4]:

```
import math
b=math.sqrt(9)
print(b)
```

```
3.0
```

Doing the maths

Function	Purpose
ceil	Returns the integer greater than or equal to the passed number
floor	Returns the integer less than or equal to the passed number
log	Returns the natural logarithm
exp	Returns the constant 'e' raised to the power of the passed number
sqrt	Returns the square root of the vaue passed
modf	Returns the fractional *and* integer part of a float

Example:

In [5]:

```
import math
a=6.9
b=math.modf(a)
print(b[0],b[1])
```

```
0.9000000000000004 6.0
```

Generating random numbers

An early computer-based PRNG, suggested by John von Neumann in 1946, is known as the middle-square method. The algorithm is as follows: take any number, square it, remove the middle digits of the resulting number as the "random number", then use that number as the seed for the next iteration.

For example, squaring the number "1111" yields "1234321", which can be written as "01234321", an 8-digit number being the square of a 4-digit number. This gives "2343" as the "random" number. Repeating this procedure gives "4896" as the next result, and so on.

Von Neumann used 10 digit numbers, but the process was the same.

Python has a system module *random* for generating random numbers:

Function	Purpose
random()	Returns a random number from the uniform distribution [0,1]
randint(m,n)	Returns a random integer between m and n
randchoice(*list*)	Returns a randomly chosen value from a list

Example:

In [6]:
```
import random
print(random.random())
```

```
0.9345101013939572
```

Generating random numbers

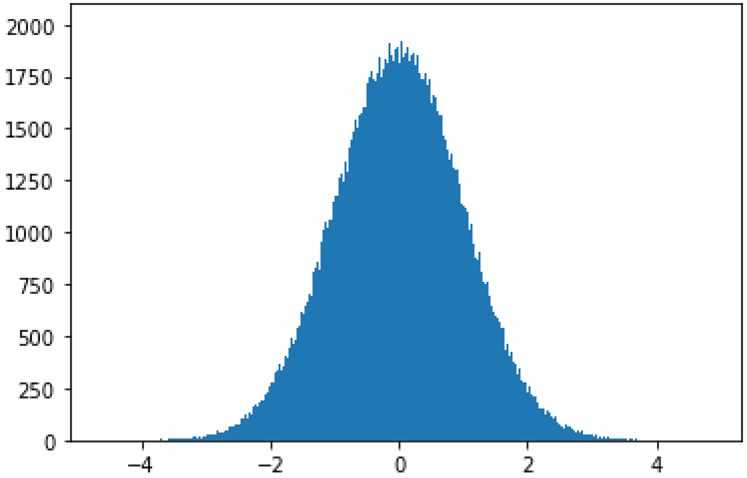
Generating random numbers using *numpy.random*:

Function	Purpose
random(size)	Draw samples of size *size* from the uniform distribution [0,1]
randn(size)	Draw samples of size *size* from the standard normal distribution
beta(a, b, size)	Draw samples from a Beta distribution
binomial(n, p, size)	Draw samples from a binomial distribution
chisquare(df[, size])	Draw samples from a chi-square distribution
dirichlet(alpha[, size])	Draw samples from the Dirichlet distribution
exponential([, scale, size])	Draw samples from an exponential distribution
f(dfnum, dfden[, size])	Draw samples from an F distribution
gamma(shape[, scale, size])	Draw samples from a Gamma distribution

Example:

In [7]:
```
import numpy.random as nr
import matplotlib.pyplot as plt
prng = nr.randn(1000000)
print(prng)
plt.hist(prng, bins=2000)
plt.show()
```

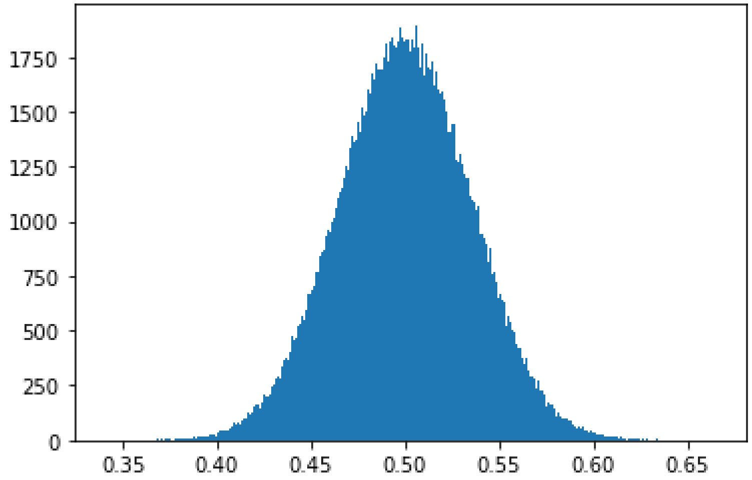
```
[0.74499448 0.85158557 0.05472761 ... 0.64896817 0.16067677 0.00941994]
```



Another Example:

In [8]:
```
import numpy.random as nr
import matplotlib.pyplot as plt
prng = nr.beta(100, 100, 1000000)
plt.hist(prng, bins=2000)
```

Out[8]:
```
(array([1., 0., 0., ..., 0., 1., 1.]),
 array([0.34015533, 0.34031759, 0.34047986, ..., 0.6643626 , 0.66452487,
        0.66468714]),
 <a list of 2000 Patch objects>)
```



Operating System module

Provides essential functionality for working with files and directories

Method	Description
getcwd	Returns the current working directory name
chdir	Changes the current working directory
listdir	Returns a list of all entities(either files or directories) in a given directory
mkdir	Creates a new directory

In [9]:

```
import os
print(os.getcwd())
print(os.listdir())
```

```
/Users/shic6/Documents/LSE/lectures2020/Week2
['ST445_Week2_Lecture.ipynb', 'Sentence.txt', 'ST445_wk2_class_with_solutions.
ipynb', 'ST445_Week2_Lecture.pdf', 'figs', 'data.csv', 'ST445_week2_Lecture.sl
ides.html', 'titanic.csv', 'MyArray.npy', '.ipynb_checkpoints', 'simultion.p
y', 'Harry Potter and the Sorcerer.txt', 'Are list comprehensions faster than
a for loop.md', 'HW2.ipynb', 'ST445_wk2_class.ipynb']
```

System Module

Can be used to obtain information about your Python system such as exceptions or the path of directiories used to import modules:

In [10]:

```
import sys
print(sys.path)
```

```
['/Users/shic6/Documents/LSE/lectures2020/Week2', '/opt/anaconda3/lib/python3
8.zip', '/opt/anaconda3/lib/python3.8', '/opt/anaconda3/lib/python3.8/lib-dynl
oad', '', '/opt/anaconda3/lib/python3.8/site-packages', '/opt/anaconda3/lib/py
thon3.8/site-packages/aeosa', '/opt/anaconda3/lib/python3.8/site-packages/IPyt
hon/extensions', '/Users/shic6/.ipython']
```

In [11]:
```
import sys
import math
try:
    print(math.sqrt(-1))
except:
    print(sys.exc_info())
else:
    print(-math.sqrt(1))
finally:
    print(-math.sqrt(1))
```

```
(<class 'ValueError'>, ValueError('math domain error'), <traceback object at 0
x1151c3ec0>)
-1.0
```

Python Data Types

Python has a number of built in data types you should know about:

1. Float
2. int
3. strings
4. boolean
5. Tuples
6. Lists
7. Dicts

In [12]:

```
a=4.6
print(type(a))
b=10
print(type(b))
```

```
<class 'float'>
<class 'int'>
```

In [13]:

```
b=int(a)
print(b,type(b))
print(type(float(10)))
```

```
4 <class 'int'>
<class 'float'>
```

Strings

Python has excellent string (text) processing capability. I can only touch on it here:

In [14]:
```
phrase='4 out of 3 people struggle with maths.'
print(phrase[0:3])
print(phrase.find('maths'))
print(phrase.find('english'))
print(phrase.lower())
print(phrase.upper())
```

```
4 o
32
-1
4 out of 3 people struggle with maths.
4 OUT OF 3 PEOPLE STRUGGLE WITH MATHS.
```

In [15]:
```
a=phrase.split(' ')
print(a)
print(type(a))
```

```
['4', 'out', 'of', '3', 'people', 'struggle', 'with', 'maths.']
<class 'list'>
```

Tuples and Lists

These are both arrays. Tuples are *immutable* whilst lists are *mutable*

In [16]:

```
numbers=[1.0,3.0,7.0]
for x in numbers:
    print(x)
for x in a:
    print(x)
```

```
1.0
3.0
7.0
4
out
of
3
people
struggle
with
maths.
```

In [17]:

```
numbers.append(9)
print(numbers)
numbers.pop(0)
print(numbers)
numbers.insert(0, 1.0)
print(numbers)
```

```
[1.0, 3.0, 7.0, 9]
[3.0, 7.0, 9]
[1.0, 3.0, 7.0, 9]
```

In [18]:

```
numbers1 = numbers
numbers1.pop()
print(numbers1)
print(numbers)
print(id(numbers), id(numbers1))
numbers1 = numbers.copy()
print(id(numbers), id(numbers1))
```

```
[1.0, 3.0, 7.0]
[1.0, 3.0, 7.0]
4648987136 4648987136
4648987136 4648984960
```

In [19]:

```
print(numbers)
numbers[0] = 3
print(numbers)
```

```
[1.0, 3.0, 7.0]
[3, 3.0, 7.0]
```

Tuples often appear as output of a method.

Class	Description	Immutable?
bool	Boolean value	✓
int	integer (arbitrary magnitude)	✓
float	floating-point number	✓
list	mutable sequence of objects	
tuple	immutable sequence of objects	✓
str	character string	✓
set	unordered set of distinct objects	
frozenset	immutable form of set class	✓
dict	associative mapping (aka dictionary)	

In [20]:

```
x = 10
y = x
print(x is y)
print(id(x))
x = x+1
print(x)
print(y)
print(x is y)
print(id(x))
```

```
True
4325055104
11
10
False
4325055136
```

Dictionaries

This is a powerful and more complex data structure. It is a mapping between *keys* and *values*. For example, these might be names and addresses. It is a single data structure with two dependent fields in it.

In [21]:

```
Mydict={'Steve':'London','Andrew':'Wales','Bill':'Edinburgh'}    # Create a small d
ictionary
print(Mydict)
Mydict['Claus']='Lemgo'          # Add to the dictionary
print(Mydict)
```

```
{'Steve': 'London', 'Andrew': 'Wales', 'Bill': 'Edinburgh'}
{'Steve': 'London', 'Andrew': 'Wales', 'Bill': 'Edinburgh', 'Claus': 'Lemgo'}
```

In [22]:

```
#for j in Mydict.keys():
#    print(j)
#for j in Mydict.values():
#    print(j)
for x,y in Mydict.items():
    print(x,y)
```

```
Steve London
Andrew Wales
Bill Edinburgh
Claus Lemgo
```

Dictionaries (Cont'd)

In [23]:

```
print(Mydict['Bill'])
#print(Mydict['William'])
print(Mydict.get('Bill'))
print(Mydict.get('William'))
```

```
Edinburgh
Edinburgh
None
```

Dictionaries (Cont'd)

In [24]:

```
myfile=open('Harry Potter and the Sorcerer.txt','r', encoding="ISO-8859-1")
dic = {}
for line in myfile:
    words = line.split()
    for w in words:
        w = w.rstrip(' ;-,.?!"')
        w = w.lstrip(' ;-,.?!"')
        w = w.lower()
        if w in dic:
            dic[w] = dic[w]+1
        else:
            dic[w] = 1
#    print(words)
myfile.close()
# print(dic)
# import operator
# sorted_dict = sorted(dic.items(), key=operator.itemgetter(1), reverse=True)
# print(sorted_dict)
```

List comprehensions

A cool feature for filtering a list according to some simple logic

zip use this to pair up lists to create a list of tuples.

"Python for Data Analysis" by Wes McKinney has more detail.

In [25]:
```
## [expression for item in list if conditional]
```

This is equivalent to:

In [26]:
```
## for item in list:
##     if conditional:
##         expression
```

List comprehensions (Cont'd)

In [27]:

```
squares = []
for x in range(10):
    squares.append(x**2)
print(squares)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

In [28]:

```
squares = [x**2 for x in range(10)]
print(squares)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Are list comprehensions faster than for loops?

https://stackoverflow.com/questions/22108488/are-list-comprehensions-and-functional-functions-faster-than-for-loops (https://stackoverflow.com/questions/22108488/are-list-comprehensions-and-functional-functions-faster-than-for-loops)

In general, using list comprehensions are a little bit faster than using loops.

In [29]:

```python
import datetime

def time_it(func, numbers, *args):
    start_t = datetime.datetime.now()
    for i in range(numbers):
        func(args[0])
    print (datetime.datetime.now()-start_t)

def square_list1(numbers):
    a = []
    for i in numbers:
        a.append(i**2)
    return a

def square_list2(numbers):
    return [i**2 for i in numbers]

def square_list3(numbers):
    a = map(lambda x: x**2, numbers)
    return a

time_it(square_list1, 10000, range(100))
time_it(square_list2, 10000, range(100))
time_it(square_list3, 10000, range(100))
```

```
0:00:00.403749
0:00:00.303971
0:00:00.002664
```

NumPy

1. NumPy is the key package for numerical computing in Python.
2. It allows you to perform operations on whole blocks of data without writing loops.
3. It includes powerful basic mathematical routines such as matrix diagonalisation and inversion.
4. You can read and write numerical values direct to disk with NumPy.
5. The fast numerical processing it permits makes it the basis of most visualisation modules.
6. More sophisticated scientific and machine learning modules are built using NumPy.

Simple Example of NumPy

In the example below np.sqrt is a *vectorised* calculation on a NumPy array set up using *arange*

In [30]:

```
import numpy as np
rng=np.arange(10)
print(rng)
print(type(rng))
print(np.sqrt(rng))
```

```
[0 1 2 3 4 5 6 7 8 9]
<class 'numpy.ndarray'>
[0.         1.         1.41421356 1.73205081 2.         2.23606798
 2.44948974 2.64575131 2.82842712 3.        ]
```

Defining a NumPy ndarray by hand

In [31]:

```
import numpy as np
a=np.array([[1.0,2.0,4.0],[-1.0,2.0,-5.0]])
print(a.shape)
print(a)
```

```
(2, 3)
[[ 1.  2.  4.]
 [-1.  2. -5.]]
```

In [32]:

```
b=np.transpose(a)
print(b.shape)
print(b)
```

```
(3, 2)
[[ 1. -1.]
 [ 2.  2.]
 [ 4. -5.]]
```

NumPy ndarray methods

NumPy provides some handy methods to summarise the data. More generally though it is better to do this using Pandas.

In [33]:
```
print(a.sum(axis=0))
```

```
[ 0.  4. -1.]
```

In [34]:
```
print(a.sum(axis=1))
print(type(a.sum(axis=1)))
```

```
[ 7. -4.]
<class 'numpy.ndarray'>
```

In [35]:
```
print(a.sum())
```

```
3.0
```

NumPy ndarray methods

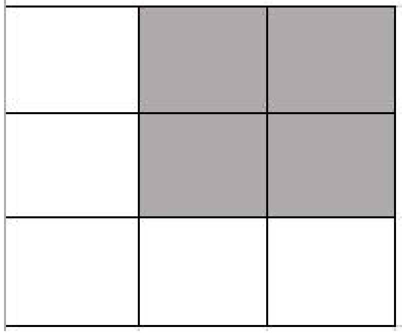
NumPy can be used for matrix calculations:

In [36]:
```
c=np.dot(a,b)
print(c)
```

```
[[ 21. -17.]
 [-17.  30.]]
```

NumPy Slicing

NumPy has a great way for selecting a subset of your dataset. The diagram below should help to explain:



Expresssion	Shape
arr[:2,1:]	(2,2)

In [37]:
```
A=np.array([[1,2,3],[4,5,6],[7,8,9]])
print(A)
print(A[0:2,1:3])
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[[2 3]
 [5 6]]
```

Boolean Indexing

You can set values in a NumPy array with simple boolean expressions:

In [38]:
```
import numpy.random as nr
arr=nr.randn(3,3)
print(arr)
```

```
[[ 1.02920931  0.16115978 -0.67492511]
 [ 0.05473878 -2.28838726  0.75880954]
 [ 1.07412868 -1.23135461 -0.74953353]]
```

In [39]:
```
import numpy as np
#print([arr<0])
#print(arr[arr<0])
arr[arr<0]=0
print(arr)
print(np.sum((arr<0)&(arr>1)))
print(np.sum((arr<0)|(arr>1)))
```

```
[[1.02920931 0.16115978 0.        ]
 [0.05473878 0.         0.75880954]
 [1.07412868 0.         0.        ]]
0
2
```

File Input and Output with ndarrays

You can save Numpy arrays directly to disk if you wish:

In [40]:
```
import numpy.random as nr
arr=nr.randn(1000,1000)
np.save('MyArray',arr)
Array=np.load('MyArray.npy')
print(Array)
```

```
[[-0.23788697  0.41199504  1.00293165 ...  0.77832032  0.27177018
  -0.3625362 ]
 [ 1.12465778  0.14443398  2.13761657 ... -2.19041312 -0.02529479
  -0.32256132]
 [-0.77548241  0.40138334  0.54533971 ... -0.42154092  0.55156245
   0.58869207]
 ...
 [ 1.7165504  -0.38128825 -0.30680085 ...  0.46689941 -0.73511793
  -0.2161228 ]
 [-0.16032632 -1.33789571 -2.03125454 ...  1.90045712  0.33572257
   0.70759576]
 [-1.30613173  0.84058191  0.7322343  ...  1.03072482 -0.37060139
   0.36510951]]
```

This kind of output can be convenient sometime but it's more common to do file Input and Outut using Pandas.

Linear Algebra

Although most of Python's mathematical capabilities are contained in specialist modules such as SciPy, Statsmodels and scikit-learn NumPy itself has some very useful methods. They are implemented using fast industry standard libraries so don't underestimate them.

1. Matrix inversion
2. Least squares
3. Matrix diagonalisation
4. Eigenvalue decomposition

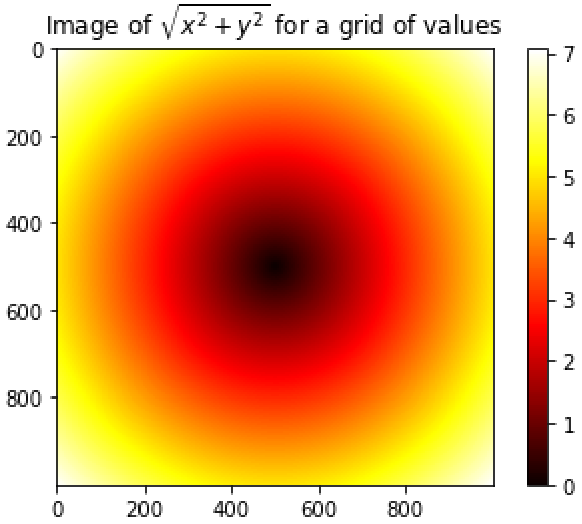
Visualising an array

(Thanks to Wes McKinney for the code)

In [41]:

```
import matplotlib.pyplot as plt
import numpy as np
points=np.arange(-5,5,0.01)
#print(points)
xs,ys=np.meshgrid(points,points)
##print(np.meshgrid(points,points))
z=np.sqrt(xs**2+ys**2)
print(np.size(z))
plt.imshow(z,cmap="hot");plt.colorbar()
plt.title("Image of $\sqrt{x^2+y^2}$ for a grid of values");plt.show()
```

```
1000000
```



Dates and Time

Date and time can be a confusing subject. Python has a *datetime* module that provides a lot of functionality and datetime objects. NumPy has its own class called datetime64.

Dates can often be imported as strings and may need to be converted to a datetime object if you want to exploit methods associated with these objects such as month or timedelta.

In [42]:

```
import datetime as dt
import numpy as np
n=dt.datetime.now()          # the current date and time
print(n)
print(n.month)               # extracts the month from the datetime object
nd=dt.datetime.date(n)       # removes the time information from te datetime object
print(nd)
```

```
2020-10-03 17:20:28.633121
10
2020-10-03
```

Dates and Time

1. You can convert datetime objects to strings and apply a format. For example 'Y' indicates a four digit year whilst 'y' indicates a two digit year.

2. You can also take a date as a string (for example imported data) and convert to a dateime object using strptime

In [43]:

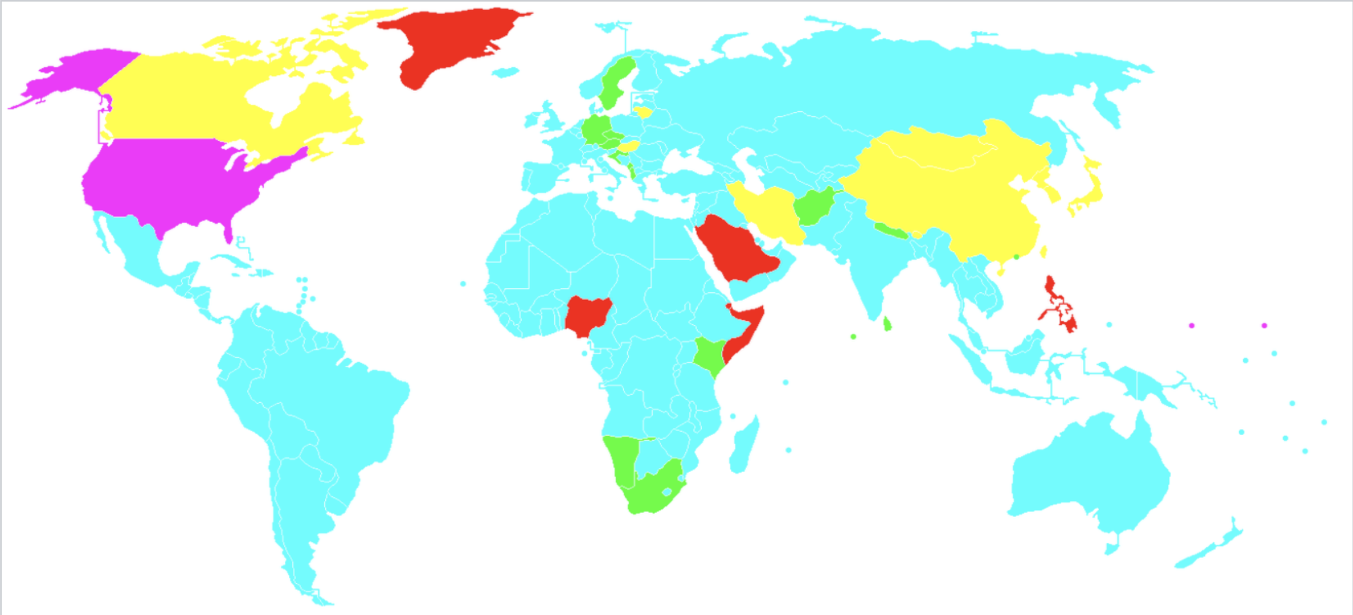
```
ds=n.strftime('%Y-%m-%d')
print(ds)
print(type(ds))
```

```
2020-10-03
<class 'str'>
```

In [44]:

```
b='10/6/2018'
print(type(b))
bd=dt.datetime.strptime(b,'%d/%m/%Y')
print(type(bd))
```

```
<class 'str'>
<class 'datetime.datetime'>
```



Color	Order styles	Main regions and countries (approximate population of each region in millions)	Approximate population in millions
Cyan	DMY	Asia (Central, SE, West), Australia (25), New Zealand (5), parts of Europe (c. 640), Latin America (625), North Africa (195), India (1315), Indonesia (265), Bangladesh (165), Russia (145)	3565
Yellow	YMD	Bhutan, Canada (35), China (1385), Koreas (75), Taiwan (24), Hungary (10), Iran (80), Japan (125), Lithuania (5), Mongolia (5). Known in other countries due to ISO 8601.	1745
Magenta	MDY	United States (325), Federated States of Micronesia, Marshall Islands	325
Red	DMY, MDY	Malaysia (35), Nigeria (190), Philippines (105), Saudi Arabia (35), Somalia (10)	380
Green	DMY, YMD	Afghanistan (28), Albania (3), Austria (9), Czech Republic (11), Kenya (49), Macau (1), Maldives, Montenegro, Namibia (2), Nepal (29), Singapore (6), South Africa (56), Sri Lanka (21), Sweden (10)[1]	225

Coming soon…

Lab with exercises on flow control in Python

Lecture next week on Pandas covering heterogeneous data structures, csv file import and basic charting.