03ex_Numpy

March 14, 2020

0.0.1 Numpy basics

1. Find the row, column and overall means for the following matrix:

```
m = np.arange(12).reshape((3,4))
```

```
In [1]: import numpy as np
        #dovrebbe essre una matrice 3x4
        m = np.arange(12).reshape((3,4))
        print (m)

        print("Row mean =    ", m.mean(axis=1))
        print("Column mean = ", m.mean(axis=0))
        print("Overall mean =", m.mean())
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
Row mean =     [1.5 5.5 9.5]
Column mean =  [4. 5. 6. 7.]
Overall mean = 5.5
```

2. Find the outer product of the following two vecotrs

```
u = np.array([1,3,5,7])
v = np.array([2,4,6,8])
```

Do this in the following ways:

- Using the function outer in numpy
- Using a nested for loop or list comprehension
- Using numpy broadcasting operatoins

```
In [2]: u = np.array([1,3,5,7])
        v = np.array([2,4,6,8])


        def outProd (x,y):
```

```
            z=np.array([x * a for a in y])
            # z=np.array([[a*b for a in x] for b in y])
            return z

        print (np.outer(u,v))
        print (outProd(u,v))
        print (np.tile(u, (4,1))*v.reshape(4,1))
        print (u*v.reshape(4,1))
```

```
[[ 2  4  6  8]
 [ 6 12 18 24]
 [10 20 30 40]
 [14 28 42 56]]
[[ 2  6 10 14]
 [ 4 12 20 28]
 [ 6 18 30 42]
 [ 8 24 40 56]]
[[ 2  6 10 14]
 [ 4 12 20 28]
 [ 6 18 30 42]
 [ 8 24 40 56]]
[[ 2  6 10 14]
 [ 4 12 20 28]
 [ 6 18 30 42]
 [ 8 24 40 56]]
```

3. Create a 10 by 6 matrix of random uniform numbers. Set all rows with any entry less than 0.1 to be zero

Hint: Use the following numpy functions - np.random.random, np.any as well as Boolean indexing and the axis argument.

```
In [3]: import numpy.random as npr

        a=np.random.random(60)
        m=a.reshape(10,6)
        print(m.round(2))
        print("")
        m[m<0.1]=0
        print(m.round(2))
```

```
[[0.45 0.69 0.55 0.99 0.23 0.99]
 [0.37 0.49 0.17 0.37 0.68 0.96]
 [0.46 0.83 0.84 0.2  0.06 0.94]
 [0.98 0.69 0.87 0.18 0.15 0.78]
 [0.76 0.77 0.74 0.7  0.58 0.8 ]
 [0.9  0.08 0.32 0.32 0.21 0.49]
 [0.73 0.72 0.73 0.53 0.69 0.56]
 [0.81 0.78 0.28 0.62 0.85 0.23]
```

```
 [0.12 0.32 0.22 0.66 0.33 0.82]
 [0.47 0.91 0.14 0.68 0.8  0.92]]

[[0.45 0.69 0.55 0.99 0.23 0.99]
 [0.37 0.49 0.17 0.37 0.68 0.96]
 [0.46 0.83 0.84 0.2  0.   0.94]
 [0.98 0.69 0.87 0.18 0.15 0.78]
 [0.76 0.77 0.74 0.7  0.58 0.8 ]
 [0.9  0.   0.32 0.32 0.21 0.49]
 [0.73 0.72 0.73 0.53 0.69 0.56]
 [0.81 0.78 0.28 0.62 0.85 0.23]
 [0.12 0.32 0.22 0.66 0.33 0.82]
 [0.47 0.91 0.14 0.68 0.8  0.92]]
```

4. Use np.linspace to create an array of 100 numbers between 0 and 2 (includsive).

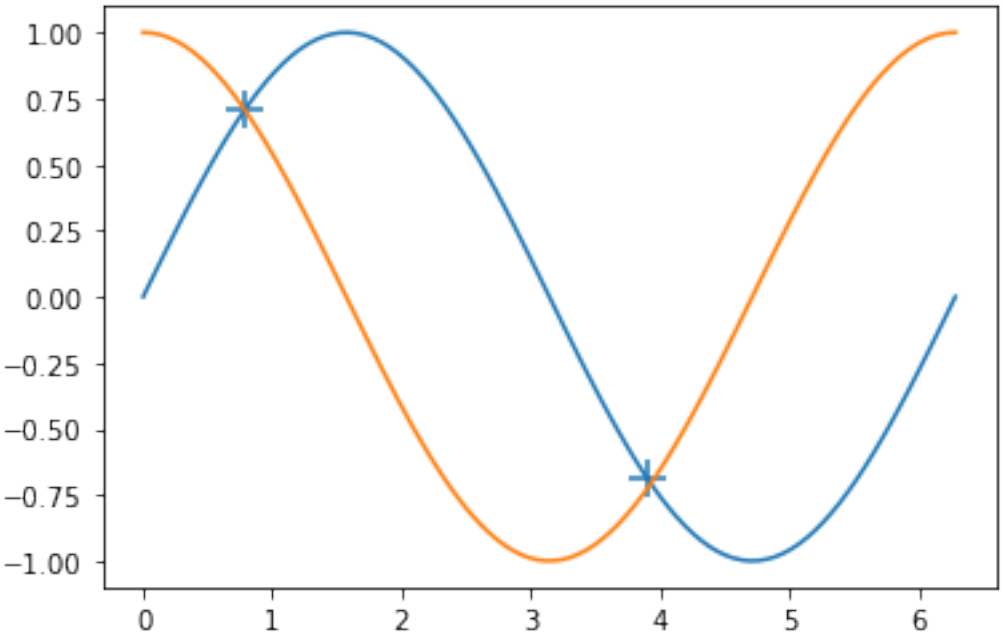
- Extract every 10th element using slice notation
- Reverse the array using slice notation
- Extract elements where the absolute difference between the sine and cosine functions evaluated at that element is less than 0.1
- Make a plot showing the sin and cos functions and indicate where they are close

```
In [4]: import matplotlib.pyplot as plt

        import math
        a=np.linspace(0, 2*np.pi, 100)
        print (a)

        b=a[::9]
        print("")
        print("10th element")
        print(b)

        c=b[::-1]
        print("")
        print("reverse")
        print(c)

        d=a[[y for y in range(0,len(a)) if abs(np.sin(a[y])-np.cos(a[y]))<0.1]]
        print("")
        print("difference eval")
        print(d)

        %matplotlib inline
        e=[]
        for i in range (0,len(d),2):
            e.append((d[i]+d[i+1])/2) #punto medio
```

```
        plt.plot(a, np.sin(a))
        plt.plot(a, np.cos(a))
        plt.scatter(e, np.sin(e), marker="+", s=200)
```

```
[0.         0.06346652 0.12693304 0.19039955 0.25386607 0.31733259
 0.38079911 0.44426563 0.50773215 0.57119866 0.63466518 0.6981317
 0.76159822 0.82506474 0.88853126 0.95199777 1.01546429 1.07893081
 1.14239733 1.20586385 1.26933037 1.33279688 1.3962634  1.45972992
 1.52319644 1.58666296 1.65012947 1.71359599 1.77706251 1.84052903
 1.90399555 1.96746207 2.03092858 2.0943951  2.15786162 2.22132814
 2.28479466 2.34826118 2.41172769 2.47519421 2.53866073 2.60212725
 2.66559377 2.72906028 2.7925268  2.85599332 2.91945984 2.98292636
 3.04639288 3.10985939 3.17332591 3.23679243 3.30025895 3.36372547
 3.42719199 3.4906585  3.55412502 3.61759154 3.68105806 3.74452458
 3.8079911  3.87145761 3.93492413 3.99839065 4.06185717 4.12532369
 4.1887902  4.25225672 4.31572324 4.37918976 4.44265628 4.5061228
 4.56958931 4.63305583 4.69652235 4.75998887 4.82345539 4.88692191
 4.95038842 5.01385494 5.07732146 5.14078798 5.2042545  5.26772102
 5.33118753 5.39465405 5.45812057 5.52158709 5.58505361 5.64852012
 5.71198664 5.77545316 5.83891968 5.9023862  5.96585272 6.02931923
 6.09278575 6.15625227 6.21971879 6.28318531]

10th element
[0.         0.57119866 1.14239733 1.71359599 2.28479466 2.85599332
 3.42719199 3.99839065 4.56958931 5.14078798 5.71198664 6.28318531]

reverse
[6.28318531 5.71198664 5.14078798 4.56958931 3.99839065 3.42719199
 2.85599332 2.28479466 1.71359599 1.14239733 0.57119866 0.        ]

difference eval
[0.76159822 0.82506474 3.87145761 3.93492413]
```

Out[4]: <matplotlib.collections.PathCollection at 0x7efbbb13bc18>



5. Create a matrix that shows the 10 by 10 multiplication table.

- Find the trace of the matrix
- Extract the anto-diagonal (this should be `array([10, 18, 24, 28, 30, 30, 28, 24, 18, 10])`)
- Extract the diagnoal offset by 1 upwards (this should be `array([ 2,  6, 12, 20, 30, 42, 56, 72, 90])`)

```
In [5]: a=np.array([[x*y for x in range (1,11)]for y in range (1,11)])
        print (a)
        print("trace", np.sum([a[i][i]for i in range(len(a))]))
        print("anti diag" ,a[9-np.arange(10), np.arange(10)])
        print("uppuer diag", a[np.arange(1,10), np.arange(0,9)])
```

```
[[  1   2   3   4   5   6   7   8   9  10]
 [  2   4   6   8  10  12  14  16  18  20]
 [  3   6   9  12  15  18  21  24  27  30]
 [  4   8  12  16  20  24  28  32  36  40]
 [  5  10  15  20  25  30  35  40  45  50]
 [  6  12  18  24  30  36  42  48  54  60]
 [  7  14  21  28  35  42  49  56  63  70]
 [  8  16  24  32  40  48  56  64  72  80]
 [  9  18  27  36  45  54  63  72  81  90]
 [ 10  20  30  40  50  60  70  80  90 100]]
trace 385
anti diag [10 18 24 28 30 30 28 24 18 10]
uppuer diag [ 2  6 12 20 30 42 56 72 90]
```

6. Use broadcasting to create a grid of distances

Route 66 crosses the following cities in the US: Chicago, Springfield, Saint-Louis, Tulsa, Oklahoma City, Amarillo, Santa Fe, Albuquerque, Flagstaff, Los Angeles The corresponding positions in miles are: 0, 198, 303, 736, 871, 1175, 1475, 1544, 1913, 2448

- Construct a 2D grid of distances among each city along Route 66
- Convert that in km (those savages...)

```
In [6]: pos=[0,198, 303, 736, 871, 1175, 1475, 1544, 1913, 2448]
        grid=np.array([abs(ip-jp) for ip in pos for jp in pos]).reshape((10,10))
        print("savage grid:", grid)
        grid=(grid*1.60934).round(1)
        print("correct grid:", grid)
```

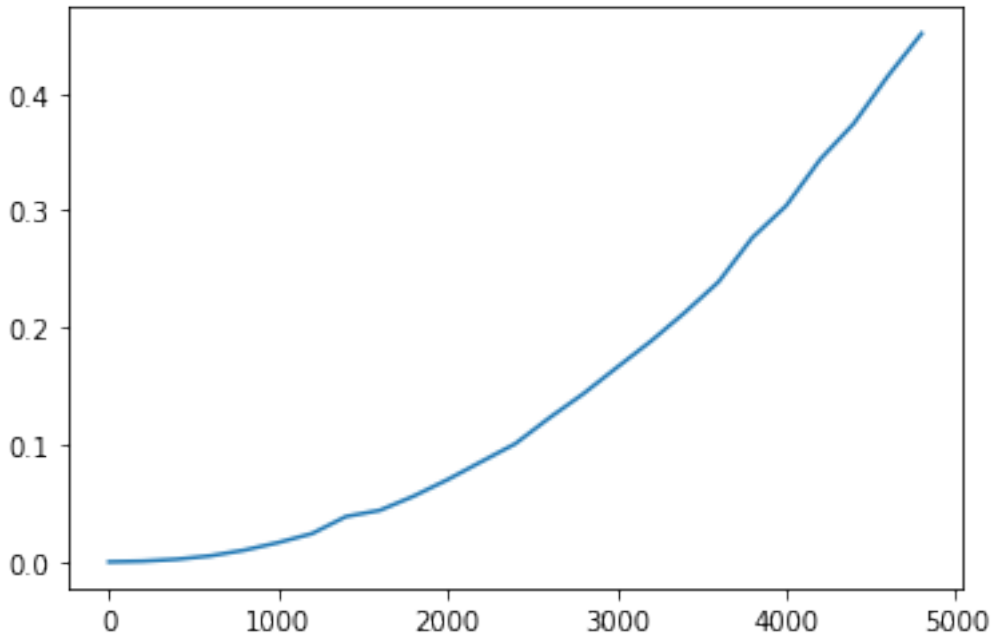
```
savage grid: [[   0  198  303  736  871 1175 1475 1544 1913 2448]
 [ 198    0  105  538  673  977 1277 1346 1715 2250]
 [ 303  105    0  433  568  872 1172 1241 1610 2145]
 [ 736  538  433    0  135  439  739  808 1177 1712]
 [ 871  673  568  135    0  304  604  673 1042 1577]
 [1175  977  872  439  304    0  300  369  738 1273]
 [1475 1277 1172  739  604  300    0   69  438  973]
 [1544 1346 1241  808  673  369   69    0  369  904]
 [1913 1715 1610 1177 1042  738  438  369    0  535]
 [2448 2250 2145 1712 1577 1273  973  904  535    0]]
correct grid: [[   0.   318.6  487.6 1184.5 1401.7 1891.  2373.8 2484.8 3078.7 3939.7]
 [ 318.6    0.   169.   865.8 1083.1 1572.3 2055.1 2166.2 2760.  3621. ]
 [ 487.6  169.     0.   696.8  914.1 1403.3 1886.1 1997.2 2591.  3452. ]
 [1184.5  865.8  696.8    0.   217.3  706.5 1189.3 1300.3 1894.2 2755.2]
 [1401.7 1083.1  914.1  217.3    0.   489.2  972.  1083.1 1676.9 2537.9]
 [1891.  1572.3 1403.3  706.5  489.2    0.   482.8  593.8 1187.7 2048.7]
 [2373.8 2055.1 1886.1 1189.3  972.   482.8    0.   111.   704.9 1565.9]
 [2484.8 2166.2 1997.2 1300.3 1083.1  593.8  111.     0.   593.8 1454.8]
 [3078.7 2760.  2591.  1894.2 1676.9 1187.7  704.9  593.8    0.   861. ]
 [3939.7 3621.  3452.  2755.2 2537.9 2048.7 1565.9 1454.8  861.     0. ]]
```

7. Prime numbers sieve: compute the prime numbers in the 0-N (N=99 to start with) range with a sieve (mask). * Constract a shape (100,) boolean array, the mask * Identify the multiples of each number starting from 2 and set accordingly the corresponding mask element * Apply the mask to obtain an array of ordered prime numbers * Check the performances (timeit); how does it scale with N? * Implement the optimization suggested in the sieve of Eratosthenes

```
In [7]: import time
        import matplotlib.pyplot as plt

        def Mask(n):
            m=np.full(n, 1, dtype=bool)
            for i in range (2,len(m)):
                prime=True
```

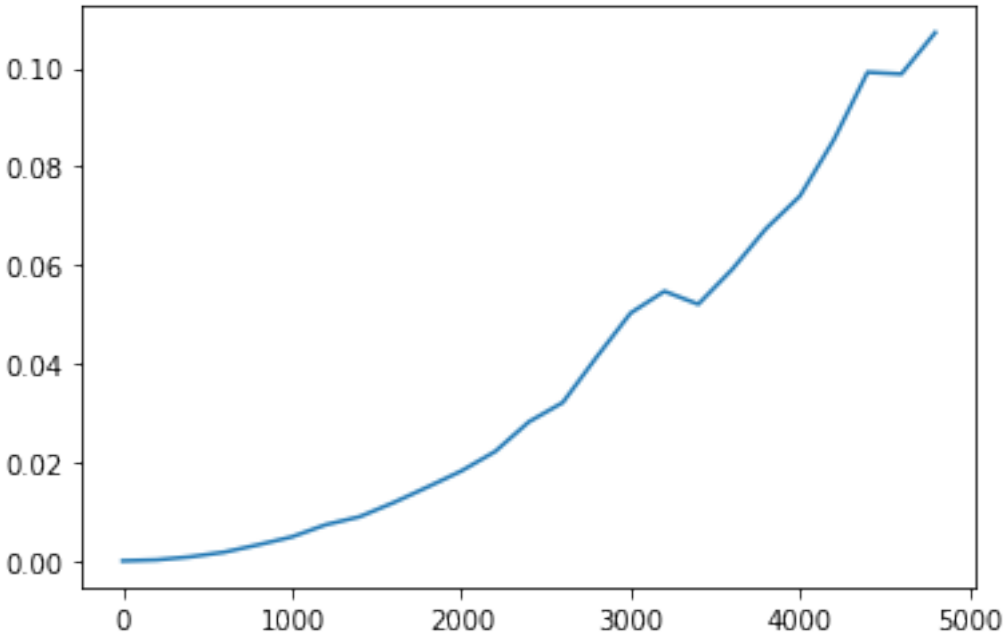
```
            for j in range (2,i):
                if i%j==0:
                    prime=False
            m[i]=prime
        return m

    m=Mask(100)
    l=np.arange(0, 100, 1)
    primes=l[m]
    print("Primes: ", primes)

    times=np.array([])
    for i in range (0, 5000, 200):
        start=time.time()
        m=Mask(i)
        stop=time.time()
        times=np.append(times,stop-start)

    %matplotlib inline
    plt.plot(np.arange(0,5000,200), times)

    print("essendo un algoritmo n^2 cresce come un polinomio di secondo grado come mi aspet
```

```
Primes:  [ 0  1  2  3  5  7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79
 83 89 97]
essendo un algoritmo n^2 cresce come un polinomio di secondo grado come mi aspetto
```



```
In [8]: def Sieve (n):
            mults= []
            primes=[1]
            for i in range(2, n+1):
                if i not in mults:
                    primes.append(i)
                    for j in range (i**2, n+1, i):
                        mults.append(j)
            return np.array(primes)

        print("Sieve primes: ", Sieve(1000))

        times=np.array([])
        for i in range (0, 5000, 200):
            start=time.time()
            m=Sieve(i)
            stop=time.time()
            times=np.append(times,stop-start)

        %matplotlib inline
        plt.plot(np.arange(0,5000,200), times)

        print("Cresce allo stesso modo ma ci mette molto meno")
```

```
Sieve primes:  [  1   2   3   5   7  11  13  17  19  23  29  31  37  41  43  47  53  59
  61  67  71  73  79  83  89  97 101 103 107 109 113 127 131 137 139 149
 151 157 163 167 173 179 181 191 193 197 199 211 223 227 229 233 239 241
 251 257 263 269 271 277 281 283 293 307 311 313 317 331 337 347 349 353
 359 367 373 379 383 389 397 401 409 419 421 431 433 439 443 449 457 461
 463 467 479 487 491 499 503 509 521 523 541 547 557 563 569 571 577 587
 593 599 601 607 613 617 619 631 641 643 647 653 659 661 673 677 683 691
 701 709 719 727 733 739 743 751 757 761 769 773 787 797 809 811 821 823
 827 829 839 853 857 859 863 877 881 883 887 907 911 919 929 937 941 947
 953 967 971 977 983 991 997]
Cresce allo stesso modo ma ci mette molto meno
```



8. Diffusion using random walk

Consider a simple random walk process: at each step in time, a walker jumps right or left (+1 or -1) with equal probability. The goal is to find the typical distance from the origin of a random walker after a given amount of time. To do that, let's simulate many walkers and create a 2D array with each walker as a raw and the actual time evolution as columns

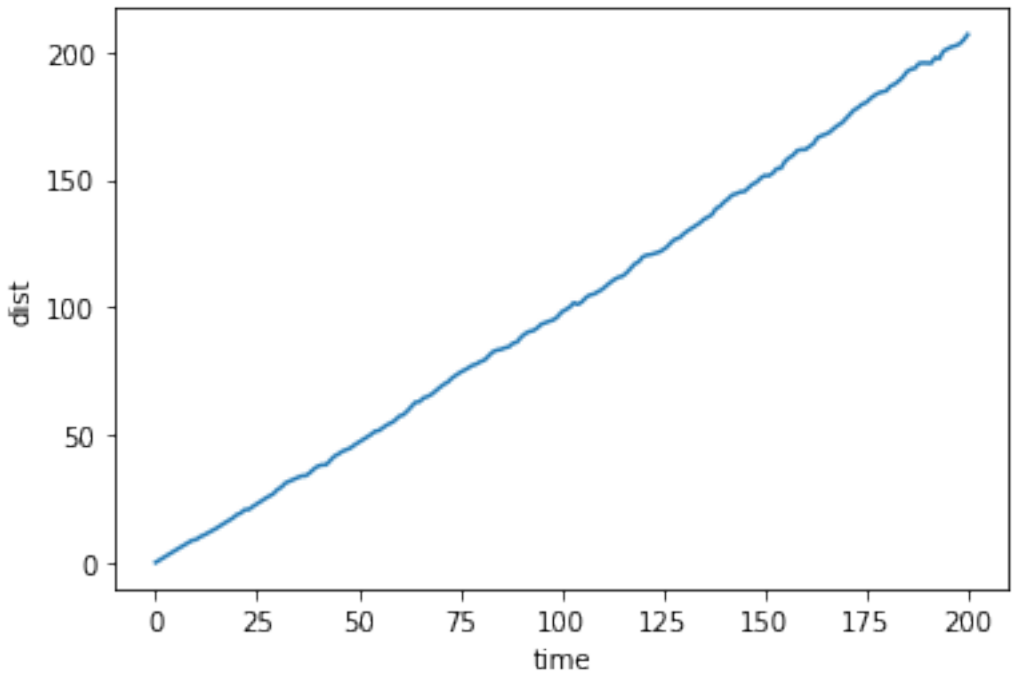
- Take 1000 walkers and let them walk for 200 steps
- Use randint to create a 2D array of size walkers x steps with values -1 or 1
- Build the actual walking distances for each walker (i.e. another 2D array "summing on each raw")
- Take the square of that 2D array (elementwise)
- Compute the mean of the squared distances at each step (i.e. the mean along the columns)
- Plot the average distances (sqrt(distance**2)) as a function of time (step)

Did you get what you expected?

```
In [9]: rnd_wlk=1000
        steps=200
        a=np.random.randint(2,size=(rnd_wlk,steps))
        #print(a)
        dist= np.zeros((rnd_wlk,steps+1))
        np.place(a, a == 0, -1)
        for i in range(a.shape[1]):
            dist[:,i+1] = dist[:,i]+a[:,i]
        #print( dist)
        square = dist**2
        mean = square.mean(axis=0)
```

```
        #print (meancols.round(1))
        time=np.arange(201)
        plt.plot(time, mean)
        plt.xlabel('time')
        plt.ylabel('dist')
```

```
Out[9]: Text(0, 0.5, 'dist')
```



9. Analyze a data file * Download the population of hares, lynxes and carrots at the beginning of the last century. `python ! wget https://www.dropbox.com/s/3vigxoqayo389uc/populations.txt`

- Check the content by looking within the file
- Load the data (use an appropriate numpy method) into a 2D array
- Create arrays out of the columns, the arrays being (in order): *year*, *hares*, *lynxes*, *carrots*
- Plot the 3 populations over the years
- Compute the main statistical properties of the dataset (mean, std, correlations, etc.)
- Which species has the highest population each year?

Do you feel there is some evident correlation here? Studies tend to believe so.

```
In [10]: ! wget https://www.dropbox.com/s/3vigxoqayo389uc/populations.txt
```

```
--2020-03-14 14:17:49--  https://www.dropbox.com/s/3vigxoqayo389uc/populations.txt
Resolving www.dropbox.com (www.dropbox.com)... 162.125.69.1, 2620:100:6025:1::a27d:4501
Connecting to www.dropbox.com (www.dropbox.com)|162.125.69.1|:443... connected.
```

```
HTTP request sent, awaiting response... 301 Moved Permanently
Location: /s/raw/3vigxoqayo389uc/populations.txt [following]
--2020-03-14 14:17:51--  https://www.dropbox.com/s/raw/3vigxoqayo389uc/populations.txt
Reusing existing connection to www.dropbox.com:443.
HTTP request sent, awaiting response... 302 Found
Location: https://ucbf70c99baed6e99a120c67fd25.dl.dropboxusercontent.com/cd/0/inline/Az5drprWd
--2020-03-14 14:17:52--  https://ucbf70c99baed6e99a120c67fd25.dl.dropboxusercontent.com/cd/0/i
Resolving ucbf70c99baed6e99a120c67fd25.dl.dropboxusercontent.com (ucbf70c99baed6e99a120c67fd25
Connecting to ucbf70c99baed6e99a120c67fd25.dl.dropboxusercontent.com (ucbf70c99baed6e99a120c67
HTTP request sent, awaiting response... 200 OK
Length: 525 [text/plain]
Saving to: populations.txt.2

populations.txt.2   100%[===================>]     525  --.-KB/s    in 0s

2020-03-14 14:17:54 (167 MB/s) - populations.txt.2 saved [525/525]
```

In [ ]: