How-to-count-distance-to-the-previous-zero

For each value, count the difference of the distance from the previous zero (or the start of the Series, whichever is closer) and if there are no previous zeros,print the position Consider a DataFrame df where there is an integer column {'X':[7, 2, 0, 3, 4, 2, 5, 0, 3, 4]} The values should therefore be [1, 2, 0, 1, 2, 3, 4, 0, 1, 2]. Make this a new column 'Y'

In [1]:
```
import numpy as np
import pandas as pd
df = pd.DataFrame({'X': [7, 2, 0, 3, 4, 2, 5, 0, 3, 4]})
print(df)
x = (df['X'] != 0).cumsum()
y = x != x.shift()
df['Y'] = y.groupby((y != y.shift()).cumsum()).cumsum()
print(df['Y'])
```

```
   X
0  7
1  2
2  0
3  3
4  4
5  2
6  5
7  0
8  3
9  4
0    1.0
1    2.0
2    0.0
3    1.0
4    2.0
5    3.0
6    4.0
7    0.0
8    1.0
9    2.0
Name: Y, dtype: float64
```

Create a DatetimeIndex that contains each business day of 2015 and use it to index a Series of random numbers

In [2]:

```
import pandas as pd
import numpy as np
date_time = pd.date_range(start='2015-01-01', end='2015-12-31', freq='B')
s = pd.Series(np.random.rand(len(date_time)), index=date_time)
s
```

```
Out[2]: 2015-01-01    0.175776
        2015-01-02    0.442423
        2015-01-05    0.128131
        2015-01-06    0.159887
        2015-01-07    0.143653
        2015-01-08    0.237001
        2015-01-09    0.958777
        2015-01-12    0.246060
        2015-01-13    0.278533
        2015-01-14    0.660045
        2015-01-15    0.549531
        2015-01-16    0.350703
        2015-01-19    0.328491
        2015-01-20    0.878701
        2015-01-21    0.309386
        2015-01-22    0.380748
        2015-01-23    0.095156
        2015-01-26    0.164633
        2015-01-27    0.688700
        2015-01-28    0.910620
        2015-01-29    0.732147
        2015-01-30    0.314490
        2015-02-02    0.998434
        2015-02-03    0.529152
        2015-02-04    0.922754
        2015-02-05    0.437131
        2015-02-06    0.395495
        2015-02-09    0.875576
        2015-02-10    0.756211
        2015-02-11    0.028193
                        ...
        2015-11-20    0.767404
        2015-11-23    0.899328
        2015-11-24    0.227174
        2015-11-25    0.874877
        2015-11-26    0.061526
        2015-11-27    0.030721
        2015-11-30    0.941286
        2015-12-01    0.391174
        2015-12-02    0.498219
        2015-12-03    0.285510
        2015-12-04    0.774146
        2015-12-07    0.046568
        2015-12-08    0.094365
        2015-12-09    0.350631
        2015-12-10    0.750434
        2015-12-11    0.859792
        2015-12-14    0.262502
        2015-12-15    0.788069
        2015-12-16    0.058731
        2015-12-17    0.252600
        2015-12-18    0.411238
        2015-12-21    0.026120
        2015-12-22    0.682057
        2015-12-23    0.446112
        2015-12-24    0.989267
        2015-12-25    0.056888
```

```
2015-12-28    0.492869
2015-12-29    0.706136
2015-12-30    0.364880
2015-12-31    0.363489
Freq: B, Length: 261, dtype: float64
```

Find the sum of the values in s for every Wednesday

```
In [3]: s[s.index.weekday == 2].sum()
```

```
Out[3]: 24.15447520130526
```

Average For each calendar month

```
In [4]: s.resample('M').mean()
```

```
Out[4]: 2015-01-31    0.415163
        2015-02-28    0.648532
        2015-03-31    0.603862
        2015-04-30    0.432662
        2015-05-31    0.395137
        2015-06-30    0.450449
        2015-07-31    0.534108
        2015-08-31    0.410153
        2015-09-30    0.483994
        2015-10-31    0.485415
        2015-11-30    0.584988
        2015-12-31    0.432687
        Freq: M, dtype: float64
```

For each group of four consecutive calendar months in s, find the date on which the highest value occurred

```
In [5]: s.groupby(pd.Grouper(freq='4M')).idxmax()
```

```
Out[5]: 2015-01-31   2015-01-09
        2015-05-31   2015-02-02
        2015-09-30   2015-09-14
        2016-01-31   2015-12-24
        dtype: datetime64[ns]
```

```
In [ ]:
```