1 How-to-count-distance-to-the-previous-zero For each value, count the difference back to the previous zero (or the start of the Series, whichever is closer) create a new column 'Y' Consider a DataFrame df where there is an integer column 'X'

In [2]:
```
#Import the Packages
import pandas as pd
import numpy as np

#Create a Data frame with the given input.
df = pd.DataFrame({'X': [7, 2, 0, 3, 4, 2, 5, 0, 3, 4]})

# Get the Indices where the Zero is present in the array.
zeroIndices = np.r_[-1, (df['X'] == 0).nonzero()[0]]

# Create a numpy array with the posiitions of the elements
idx = np.arange(len(df))

# Find the indices into a sorted array a such that, if the corresponding elements in v were inserted before t
he indices,
# the order of a would be preserved.
df['Y'] = idx - zeroIndices[np.searchsorted(zeroIndices - 1, idx) - 1]
df
```

Out[2]:

	X	Y
0	7	1
1	2	2
2	0	0
3	3	1
4	4	2
5	2	3
6	5	4
7	0	0
8	3	1
9	4	2

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

In [4]:
```
#Import the Packages
import pandas as pd

dti = pd.date_range(start='2015-01-01', end='2015-12-31', freq='B')
s = pd.Series(np.random.rand(len(dti)), index = dti)
print(s)
```

```
2015-01-01    0.365203
2015-01-02    0.559905
2015-01-05    0.574506
2015-01-06    0.049900
2015-01-07    0.718557
2015-01-08    0.982028
2015-01-09    0.761938
2015-01-12    0.612777
2015-01-13    0.316099
2015-01-14    0.486651
2015-01-15    0.883262
2015-01-16    0.718176
2015-01-19    0.595434
2015-01-20    0.840022
2015-01-21    0.176687
2015-01-22    0.917900
2015-01-23    0.536236
2015-01-26    0.755045
2015-01-27    0.303119
2015-01-28    0.152015
2015-01-29    0.078310
2015-01-30    0.421178
2015-02-02    0.601883
2015-02-03    0.607353
2015-02-04    0.583188
2015-02-05    0.387329
2015-02-06    0.860125
2015-02-09    0.409198
2015-02-10    0.850391
2015-02-11    0.914566
                ...
2015-11-20    0.062823
2015-11-23    0.704085
2015-11-24    0.478163
2015-11-25    0.818701
2015-11-26    0.312053
2015-11-27    0.088808
2015-11-30    0.543560
2015-12-01    0.822780
2015-12-02    0.001446
2015-12-03    0.215467
2015-12-04    0.085623
2015-12-07    0.097757
```

```
2015-12-08    0.362428
2015-12-09    0.195553
2015-12-10    0.974432
2015-12-11    0.870651
2015-12-14    0.172564
2015-12-15    0.668117
2015-12-16    0.537227
2015-12-17    0.804708
2015-12-18    0.564229
2015-12-21    0.130800
2015-12-22    0.838921
2015-12-23    0.964605
2015-12-24    0.183870
2015-12-25    0.713920
2015-12-28    0.634847
2015-12-29    0.972074
2015-12-30    0.649989
2015-12-31    0.055652
Freq: B, Length: 261, dtype: float64
```

3. Find the sum of the values in s for every Wednesday.

In [5]:
```
print(s[s.index.weekday == 2].sum())
```

```
22.94827765534374
```

4. Average For each calendar month

In [6]:
```
print(s.resample('M').mean())
```

```
2015-01-31    0.536589
2015-02-28    0.604015
2015-03-31    0.480730
2015-04-30    0.500721
2015-05-31    0.475343
2015-06-30    0.452114
2015-07-31    0.507564
2015-08-31    0.554628
2015-09-30    0.420181
2015-10-31    0.541267
2015-11-30    0.502809
2015-12-31    0.500768
Freq: M, dtype: float64
```

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

In [7]:
```
print(s.groupby(pd.Grouper(freq='4M')).max())
print(s.groupby(pd.Grouper(freq='4M', closed='left')).max())
```

```
2015-01-31    0.982028
2015-05-31    0.999994
2015-09-30    0.986463
2016-01-31    0.995124
dtype: float64
2015-04-30    0.999994
2015-08-31    0.986463
2015-12-31    0.995124
2016-04-30    0.055652
Freq: 4M, dtype: float64
```