In [3]:

```python
"""
1) 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'.
import pandas as pd
df = pd.DataFrame({'X': [7, 2, 0, 3, 4, 2, 5, 0, 3, 4]})
"""

import pandas as pd
import numpy as np

# Storing the Given list as Dataframe in to df
df = pd.DataFrame({'X': [7, 2, 0, 3, 4, 2, 5, 0, 3, 4]})
#cumsum increment every time new value is encountered and stored in X when its true x will become zero again.
x = (df['X'] != 0).cumsum()
y = x != x.shift()
#Which marks every time the difference between rows is non-zero, so that I can use it to spot transitions in data
df['Y'] = y.groupby((y != y.shift()).cumsum()).cumsum()
#cumsum increment every time new value is encountered and stored in X when its true x will become zero again.
x = (df['X'] != 0).cumsum()
y = x != x.shift()
#Which marks every time the difference between rows is non-zero, so that I can use it to spot transitions in data
df['Y'] = y.groupby((y != y.shift()).cumsum()).cumsum()
# printing the result data as data frame
print(df)
```

```
   X    Y
0  7  1.0
1  2  2.0
2  0  0.0
3  3  1.0
4  4  2.0
5  2  3.0
6  5  4.0
7  0  0.0
8  3  1.0
9  4  2.0
```

In [4]:
```
"""
Create a DatetimeIndex that contains each business day of 2015 and use it to index a Series of random numbers
"""
# cfeating date time index with pandas Date range
datetimeindex = pd.date_range(start='2015-01-01', end='2015-12-31')
# created series with date time index
s = pd.Series(np.random.rand(len(datetimeindex)),index=datetimeindex)
print(s)
```

```
2015-01-01    0.380967
2015-01-02    0.987785
2015-01-03    0.199712
2015-01-04    0.722847
2015-01-05    0.410392
2015-01-06    0.324549
2015-01-07    0.557306
2015-01-08    0.626344
2015-01-09    0.905808
2015-01-10    0.870526
2015-01-11    0.486774
2015-01-12    0.256257
2015-01-13    0.893767
2015-01-14    0.879269
2015-01-15    0.357658
2015-01-16    0.966158
2015-01-17    0.014704
2015-01-18    0.679286
2015-01-19    0.664565
2015-01-20    0.761804
2015-01-21    0.333744
2015-01-22    0.634705
2015-01-23    0.571425
2015-01-24    0.297446
2015-01-25    0.639811
2015-01-26    0.208956
2015-01-27    0.903773
2015-01-28    0.027225
2015-01-29    0.352454
2015-01-30    0.790986
                ...
2015-12-02    0.661664
```

```
2015-12-03    0.964125
2015-12-04    0.023331
2015-12-05    0.502869
2015-12-06    0.992488
2015-12-07    0.668223
2015-12-08    0.271019
2015-12-09    0.009032
2015-12-10    0.969018
2015-12-11    0.809800
2015-12-12    0.641828
2015-12-13    0.909451
2015-12-14    0.434620
2015-12-15    0.304097
2015-12-16    0.432041
2015-12-17    0.147499
2015-12-18    0.170324
2015-12-19    0.082422
2015-12-20    0.719889
2015-12-21    0.377477
2015-12-22    0.910338
2015-12-23    0.904271
2015-12-24    0.985582
2015-12-25    0.090768
2015-12-26    0.959515
2015-12-27    0.469429
2015-12-28    0.938544
2015-12-29    0.577822
2015-12-30    0.553813
2015-12-31    0.711446
Freq: D, Length: 365, dtype: float64
```

In [5]:

```
"""
3) Find the sum of the values in s for every Wednesday
"""
# finding sum of the wednesdays using weekday_ name
s[datetimeindex.weekday_name == 'Wednesday'].sum()
```

Out[5]:

```
23.6029531737413
```

In [6]:
```
"""
4) Average For each calendar month
"""
# calculating average using mean by frequency as month
s.groupby(pd.Grouper(freq='M')).mean()
```

Out[6]:
```
2015-01-31    0.562474
2015-02-28    0.500987
2015-03-31    0.483855
2015-04-30    0.467110
2015-05-31    0.445663
2015-06-30    0.454396
2015-07-31    0.486156
2015-08-31    0.541759
2015-09-30    0.485145
2015-10-31    0.506159
2015-11-30    0.407536
2015-12-31    0.567620
Freq: M, dtype: float64
```

In [9]:
```
"""
5) For each group of four consecutive calendar months in s, find the date on which the highest value occurred.
"""
# calculating Highest value using max value of the month by max using frequency as quarter with is 4 quarters
s.groupby(pd.Grouper(freq='4M')).max()

```

Out[9]:
```
2015-01-31    0.987785
2015-05-31    0.997545
2015-09-30    0.985427
2016-01-31    0.992488
dtype: float64
```

In [13]:
```
# calculating Highest value date  using idmax value of the month by max using frequency as quarter with is 4 quarters
s.groupby(pd.Grouper(freq='4M')).idxmax()
```

Out[13]:
```
2015-01-31   2015-01-02
2015-05-31   2015-04-22
2015-09-30   2015-08-13
2016-01-31   2015-12-06
dtype: datetime64[ns]
```

In [ ]:
```

```