In [53]:
```
#high performace in python using eval and query
```

In [54]:
```
#numexpr package
import numexpr
a=np.array([1,2,3,4,5,6])
data=numexpr.evaluate("(a>2)&(a<5)")
print(data)
```

```
[False False  True  True False False]
```

In [73]:
```
#eval uses string expressions to efficiently compute operations on dataframe
import pandas as pd
rng=np.random.RandomState(42)
df1=(pd.DataFrame(rng.rand(10000, 100)))
df2=(pd.DataFrame(rng.rand(10000, 100)))
df3=(pd.DataFrame(rng.rand(10000, 100)))
df4=(pd.DataFrame(rng.rand(10000, 100)))
%timeit df1+df2+df3+df4
```

```
14.4 ms ± 1.13 ms per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

In [74]:
```
%timeit pd.eval('df1+df2+df3+df4')
```

```
5.97 ms ± 203 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

In [57]:
```
df1,df2,df3,df4,df5 = (pd.DataFrame(rng.randint(0, 1000, (100, 3)))for i in range(5))
result1=-df1*df2/(df3+df4)-df5
result2=pd.eval("-df1*df2/(df3+df4)-df5")
np.allclose(result1, result2)
#eval supports bitwise operators,literal and,or and also comparison operators
#eval does ot support coditional statemets,loops and fuction calls
```

Out[57]:
```
True
```

In [58]:
```
df=pd.DataFrame(rng.rand(1000,4),columns=["A","B","C","D"])
df.head()
```

Out[58]:

	A	B	C	D
0	0.756636	0.258688	0.193133	0.084098
1	0.326434	0.559844	0.115844	0.376695
2	0.009072	0.818679	0.128449	0.283343
3	0.018428	0.432241	0.840122	0.392151
4	0.179497	0.753699	0.788807	0.173753

In [59]:
```
result=pd.eval("(df.A+df.B)/(df.C-1)")
print(result.head())
#DataFrame.eval will allow more easier computations
result=df.eval("(A+B)/(C-1)")
print(result.head())
```

```
0   -1.258354
1   -1.002400
2   -0.949745
3   -2.818819
4   -4.418688
dtype: float64
0   -1.258354
1   -1.002400
2   -0.949745
3   -2.818819
4   -4.418688
dtype: float64
```

In [60]:
```
#column assignment using eval
print(df.head())
df.eval("E=((A+B)-C)*D",inplace=True)#creating new column E based on other columns using eval
#inplace value true will mutate exisitng dataframe
print(df.head())
```

```
          A         B         C         D
0  0.756636  0.258688  0.193133  0.084098
1  0.326434  0.559844  0.115844  0.376695
2  0.009072  0.818679  0.128449  0.283343
3  0.018428  0.432241  0.840122  0.392151
4  0.179497  0.753699  0.788807  0.173753
          A         B         C         D         E
0  0.756636  0.258688  0.193133  0.084098  0.069145
1  0.326434  0.559844  0.115844  0.376695  0.290219
2  0.009072  0.818679  0.128449  0.283343  0.198143
3  0.018428  0.432241  0.840122  0.392151 -0.152724
4  0.179497  0.753699  0.788807  0.173753  0.025088
```

In [61]:
```
#modifying existing column using eval
df.eval("E=D",inplace=True)
print(df.head())
```

```
          A         B         C         D         E
0  0.756636  0.258688  0.193133  0.084098  0.084098
1  0.326434  0.559844  0.115844  0.376695  0.376695
2  0.009072  0.818679  0.128449  0.283343  0.283343
3  0.018428  0.432241  0.840122  0.392151  0.392151
4  0.179497  0.753699  0.788807  0.173753  0.173753
```

In [63]:
```
#workinbg with local variables in eval
A_mean=df["A"].mean()
df.eval("F=@A_mean",inplace=True)
print(df.head())
#@is supported only by DataFrame.eval() and not by pandas.eval()
```

```
          A         B         C         D         E         F
0  0.756636  0.258688  0.193133  0.084098  0.084098  0.501289
1  0.326434  0.559844  0.115844  0.376695  0.376695  0.501289
2  0.009072  0.818679  0.128449  0.283343  0.283343  0.501289
3  0.018428  0.432241  0.840122  0.392151  0.392151  0.501289
4  0.179497  0.753699  0.788807  0.173753  0.173753  0.501289
```

In [69]:
```
#query method
result=df[(df["A"]<0.5)&(df["B"]<0.5)]
print(result.head())
result1=df.query("A<0.5 & B<0.5")#query is mainly used for masking in dataframes
print(result1.head())
#query also accepts @ symbol for local variables
```

```
           A         B         C         D         E         F
3   0.018428  0.432241  0.840122  0.392151  0.392151  0.501289
10  0.001029  0.158938  0.468539  0.944304  0.944304  0.501289
18  0.159246  0.203701  0.091164  0.153658  0.153658  0.501289
21  0.462097  0.001232  0.598327  0.607306  0.607306  0.501289
28  0.018368  0.482356  0.210350  0.346017  0.346017  0.501289
           A         B         C         D         E         F
3   0.018428  0.432241  0.840122  0.392151  0.392151  0.501289
10  0.001029  0.158938  0.468539  0.944304  0.944304  0.501289
18  0.159246  0.203701  0.091164  0.153658  0.153658  0.501289
21  0.462097  0.001232  0.598327  0.607306  0.607306  0.501289
28  0.018368  0.482356  0.210350  0.346017  0.346017  0.501289
```

In [75]:
```
#for smaller arrays traditioal methods is fast,for larger arrays use eval and query
#use eval and query if system memory is sigificant factor that is if large arrays cant be alloted memory
```