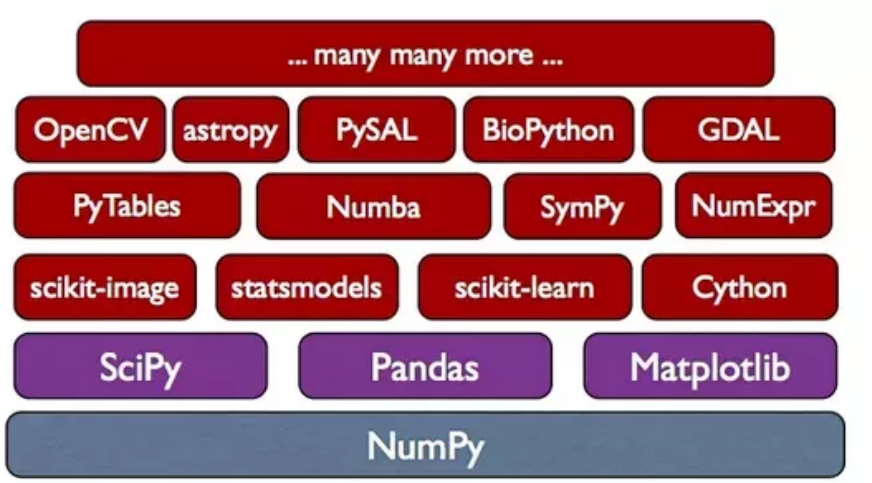
Numpy arrays

In [1]:
```
#Biblioteki do Data Science
from IPython.display import Image
Image('dependencies.png')
```

Out[1]:



In [2]:
```
import numpy as np
```

Macierze (2d) i wektory (1d): np.array

In [3]:
```
# array z listy
lista = [1,2,3]
lista_list = [[1,2,3],[4,5,6],[7,8,9]]

array_1d = np.array(lista)
array_2d = np.array(lista_list)
array_1d
# w konsoli 'from pprint import pprint' potem zamiast p
rint() pprint()
```

Out[3]:
```
array([1, 2, 3])
```

In [4]:
```
array_2d
```

Out[4]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [5]:
```
#array generowany
#arange
np.arange(0,10)
```

Out[5]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [6]:
```
np.arange(0,11,2)
```

Out[6]:
```
array([ 0,  2,  4,  6,  8, 10])
```

In [7]:
```
#zeros and ones
np.zeros(4)
```

Out[7]:
```
array([ 0.,  0.,  0.,  0.])
```

In [8]:
```
np.zeros((4,4))
```

Out[8]:
```
array([[ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.],
       [ 0.,  0.,  0.,  0.]])
```

In [9]:
```
np.ones(4)
```

Out[9]:
```
array([ 1.,  1.,  1.,  1.])
```

In [10]:
```
np.ones((4,4))
```

Out[10]:
```
array([[ 1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.],
       [ 1.,  1.,  1.,  1.]])
```

In [11]:
```
#linspace
np.linspace(0,1,11)
```

Out[11]:
```
array([ 0. ,  0.1,  0.2,  0.3,  0.4,  0.5,  0.6,  0.7,
0.8,  0.9,  1. ])
```

In [12]:
```
#eye
np.eye(5)
```

Out[12]:
```
array([[ 1.,  0.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.,  0.],
       [ 0.,  0.,  1.,  0.,  0.],
       [ 0.,  0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  0.,  1.]])
```

In [13]:
```
#random arrays
# rand - wartosci [0,1)

np.random.rand(3)
```

Out[13]:
```
array([ 0.29481885,  0.8742714 ,  0.87142848])
```

In [14]:
```
np.random.rand(3,3)
```

Out[14]:
```
array([[ 0.3220313 ,  0.34393911,  0.08458512],
       [ 0.24850104,  0.90544703,  0.40966506],
       [ 0.8937912 ,  0.6073936 ,  0.88654452]])
```

In [15]:
```
#randn - "standard normal" distribution. Unlike rand wh
ich is uniform
np.random.randn(3)
```

Out[15]:
```
array([-0.21574547, -0.3007341 ,  0.42935252])
```

In [16]:
```
np.random.randn(3,3)
```

Out[16]:
```
array([[ 0.5525676 ,  1.31060256,  1.78058706],
       [-1.09193842, -0.4450239 ,  0.14184719],
       [-0.22125783, -0.63504068,  0.28212169]])
```

In [17]:
```
# randint
# Losowe liczby calkowite z zakresu min (wlacznie) i ma
x (wylacznie)
# Trzeci argument (fakultatywny, okresla liczbe zwracan
ych liczb i ew. ich ksztalt)
np.random.randint(0,10)
```

Out[17]:
```
2
```

In [18]:
```
np.random.randint(0,10,3)
```

Out[18]:
```
array([0, 7, 3])
```

In [19]:
```
np.random.randint(0,10,(3,3))
```

Out[19]:
```
array([[8, 7, 2],
       [5, 7, 5],
       [0, 7, 6]])
```

In [20]:
```
# wlasnosci array'ow
array_1d = np.random.randint(0,100,25)
print(array_1d)
```

```
[91 67 39 70 31  5 84 85 48 14 59 26 38 16 68 11 97 44 3
0 34 34 13 80 86 94]
```

In [21]:
```
# reshape
array_1d.reshape(5,5)
```

Out[21]:
```
array([[91, 67, 39, 70, 31],
       [ 5, 84, 85, 48, 14],
       [59, 26, 38, 16, 68],
       [11, 97, 44, 30, 34],
       [34, 13, 80, 86, 94]])
```

In [22]:
```
array_1d.reshape(5,2)
```

```
---------------------------------------------------------------
-------------------
ValueError                                Traceback (mos
t recent call last)
<ipython-input-22-ab58b05a5f23> in <module>()
----> 1 array_1d.reshape(5,2)

ValueError: cannot reshape array of size 25 into shape (
5,2)
```

In [23]:
```
#wartosc maksymalna
array_1d.max()
```

Out[23]:
```
97
```

In [24]:
```
#pozycja wartosci maksymalnej (pierwsza mozliwa)
array_1d.argmax()
```

Out[24]:
```
16
```

In [25]:
```
# wartosc minimalna
array_1d.min()
```

Out[25]:
```
5
```

In [26]:
```
array_1d.argmin()
```

Out[26]:
```
5
```

In [27]:
```
# ksztalt (własnosc, nie metoda)
array_1d.shape
```

Out[27]:
```
(25,)
```

In [28]:
```
array_1d_reshaped = array_1d.reshape(5,5)
array_1d_reshaped.shape
```

Out[28]:
```
(5, 5)
```

In [29]:
```
#type of data
array_1d.dtype
```

Out[29]:
```
dtype('int64')
```

In [30]:
```
array_various = np.array([2,4,'this'])
print(array_various)
```

```
['2' '4' 'this']
```

In [32]:
```
array_various.dtype
```

Out[32]:
```
dtype('<U21')
```

In [33]:
```
#indexing and selection
array_1d
```

Out[33]:
```
array([91, 67, 39, 70, 31,  5, 84, 85, 48, 14, 59, 26, 3
8, 16, 68, 11, 97,
       44, 30, 34, 34, 13, 80, 86, 94])
```

In [34]:
```
#indexing - tak jak w listach
array_1d[4]
```

Out[34]:
```
31
```

In [35]:
```
array_1d[4:6]
```

Out[35]:
```
array([31,  5])
```

In [36]:
```
#broadcasting
```

In [37]:
```
array_1d[0:2] = 5
array_1d
```

Out[37]:
```
array([ 5,  5, 39, 70, 31,  5, 84, 85, 48, 14, 59, 26, 3
8, 16, 68, 11, 97,
       44, 30, 34, 34, 13, 80, 86, 94])
```

In [38]:
```
#Python domyslnie nie tworzy dodatkowych kopii, chyba ż
e mu to wyraznie powiemy
frag = array_1d[0:2]
frag
```

Out[38]:
```
array([5, 5])
```

In [39]:
```
frag[:] = 30
frag
```

Out[39]:
```
array([30, 30])
```

In [40]:
```
#tutaj tez nadpisane!
array_1d
```

Out[40]:
```
array([30, 30, 39, 70, 31,  5, 84, 85, 48, 14, 59, 26, 3
8, 16, 68, 11, 97,
       44, 30, 34, 34, 13, 80, 86, 94])
```

In [41]:
```
# kopiowanie
frag_copy = array_1d[0:2].copy()
frag_copy
```

Out[41]:
```
array([30, 30])
```

In [42]:
```
frag_copy[:] = 0
frag_copy
```

Out[42]:
```
array([0, 0])
```

In [43]:
```
array_1d
```

Out[43]:
```
array([30, 30, 39, 70, 31,  5, 84, 85, 48, 14, 59, 26, 3
8, 16, 68, 11, 97,
       44, 30, 34, 34, 13, 80, 86, 94])
```

In [44]:
```
#dwuwymiarowe array'e
# array[row][col] lub array[row,col]
array_2d
```

Out[44]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [45]:
```
array_2d[1]
```

Out[45]:
```
array([4, 5, 6])
```

In [46]:
```
array_2d[1,1]
```

Out[46]:
```
5
```

In [47]:
```
array_2d[:,1]
```

Out[47]:
```
array([2, 5, 8])
```

In [48]:
```
array_2d[:2, 1:]
```

Out[48]:
```
array([[2, 3],
       [5, 6]])
```

In [49]:
```
#wybrane wiersze
array_2d[[0,2]]
```

Out[49]:
```
array([[1, 2, 3],
       [7, 8, 9]])
```

In [50]:
```
#zmiana kolejnosci wybranych wierszy
array_2d[[2,0]]
```

Out[50]:
```
array([[7, 8, 9],
       [1, 2, 3]])
```

In [51]:
```
#analogicznie dla kolumn
array_2d[:,[0,2]]
```

Out[51]:
```
array([[1, 3],
       [4, 6],
       [7, 9]])
```

In [52]:
```
array_2d[[0,2], [0,2]]
```

Out[52]:
```
array([1, 9])
```

In [53]:
```
# selekcja elementow spelniajacych kryteria - rozwiazan
ie list comprehension
array_2d < 5
```

Out[53]:
```
array([[ True,  True,  True],
       [ True, False, False],
       [False, False, False]], dtype=bool)
```

In [54]:
```
array_2d[array_2d < 5]
```

Out[54]:
```
array([1, 2, 3, 4])
```

In [55]:
```
# operacje na array'ach
array_2d
```

Out[55]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [56]:
```
array_2d + array_2d
```

Out[56]:
```
array([[ 2,  4,  6],
       [ 8, 10, 12],
       [14, 16, 18]])
```

In [57]:
```
array_2d * array_2d
```

Out[57]:
```
array([[ 1,  4,  9],
       [16, 25, 36],
       [49, 64, 81]])
```

In [58]:
```
array_2d - array_2d
```

Out[58]:
```
array([[0, 0, 0],
       [0, 0, 0],
       [0, 0, 0]])
```

In [59]:
```
array_2d / array_2d
```

Out[59]:
```
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
```

In [60]:
```
array_2d = np.arange(0,9).reshape(3,3)
array_2d
```

Out[60]:
```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

In [61]:
```
array_2d/array_2d
```

```
/usr/local/lib/python3.5/dist-packages/ipykernel/__main_
_.py:1: RuntimeWarning: invalid value encountered in tru
e_divide
  if __name__ == '__main__':
```

Out[61]:
```
array([[ nan,   1.,   1.],
       [  1.,   1.,   1.],
       [  1.,   1.,   1.]])
```

In [62]:
```
1/array_2d
```

```
/usr/local/lib/python3.5/dist-packages/ipykernel/__main_
_.py:1: RuntimeWarning: divide by zero encountered in tr
ue_divide
  if __name__ == '__main__':
```

Out[62]:
```
array([[        inf,  1.        ,  0.5       ],
       [ 0.33333333,  0.25      ,  0.2       ],
       [ 0.16666667,  0.14285714,  0.125     ]])
```

In [63]:
```
array_2d**2
```

Out[63]:
```
array([[ 0,  1,  4],
       [ 9, 16, 25],
       [36, 49, 64]])
```

In [64]:
```
# rozne matematyczne funkcje
np.sqrt(array_2d)
```

Out[64]:
```
array([[ 0.        ,  1.        ,  1.41421356],
       [ 1.73205081,  2.        ,  2.23606798],
       [ 2.44948974,  2.64575131,  2.82842712]])
```

In [65]:
```
# wykladnicza
np.exp(array_2d)
```

Out[65]:
```
array([[  1.00000000e+00,   2.71828183e+00,   7.38905610
e+00],
       [  2.00855369e+01,   5.45981500e+01,   1.48413159
e+02],
       [  4.03428793e+02,   1.09663316e+03,   2.98095799
e+03]])
```

In [66]:
```
np.max(array_2d) == array_2d.max()
```

Out[66]:
```
True
```

In [67]:
```
np.sin(array_2d)
```

Out[67]:
```
array([[ 0.        ,  0.84147098,  0.90929743],
       [ 0.14112001, -0.7568025 , -0.95892427],
       [-0.2794155 ,  0.6569866 ,  0.98935825]])
```

In [68]:
```
np.log(array_2d)
```

```
/usr/local/lib/python3.5/dist-packages/ipykernel/__main_
_.py:1: RuntimeWarning: divide by zero encountered in lo
g
  if __name__ == '__main__':
```

Out[68]:
```
array([[       -inf,  0.        ,  0.69314718],
       [ 1.09861229,  1.38629436,  1.60943791],
       [ 1.79175947,  1.94591015,  2.07944154]])
```

cwiczenie 1

stworzcie macierz:

array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10], [11, 12, 13, 14, 15], [16, 17, 18, 19, 20], [21, 22, 23, 24, 25]])

uzyskajcie:

1) array([[12, 13, 14, 15], [17, 18, 19, 20], [22, 23, 24, 25]])

2) 20

3) array([[2], [7], [12]])

4) sume wszystkich elementow w macierzy

5) sume dla każdej kolumny w macierzy

Pandas

```
In [69]: import pandas as pd
```

glowna struktura danych : Series, można tworzyc z list, słownikow, array'ow

```
In [70]: lista_1 = [1,2,3]
         pd.Series(data=lista_1)
```

```
Out[70]: 0    1
         1    2
         2    3
         dtype: int64
```

```
In [71]: lista_2 = ['a', 'b', 'c']
         pd.Series(data=lista_2)
```

```
Out[71]: 0    a
         1    b
         2    c
         dtype: object
```

```
In [72]: lista_3 = [1, 2, 'c']
         pd.Series(data=lista_3)
```

```
Out[72]: 0    1
         1    2
         2    c
         dtype: object
```

In [71]:
```
pd.Series(data=lista_1, index=lista_2)
```

Out[71]:
```
a    1
b    2
c    3
dtype: int64
```

In [73]:
```
array_1 = np.array([1,2,3])
pd.Series(data=array_1)
```

Out[73]:
```
0    1
1    2
2    3
dtype: int64
```

In [74]:
```
pd.Series(data=array_1, index=lista_2)
```

Out[74]:
```
a    1
b    2
c    3
dtype: int64
```

In [75]:
```
dict_1 = {1:'a', 2:'b', 3:'c'}
pd.Series(data=dict_1)
```

Out[75]:
```
1    a
2    b
3    c
dtype: object
```

In [76]:
```
ser1 = pd.Series([0,5,1,4],index = ['rower', 'kajaki','
kite', 'basen'])
ser2 = pd.Series([1,0,5,2],index = ['rower', 'kajaki','
biegi', 'basen'])
#odwolanie po indexie
ser1['basen']
```

Out[76]:
```
4
```

In [77]:
```
ser1+ser2
```

Out[77]:
```
basen     6.0
biegi     NaN
kajaki    5.0
kite      NaN
rower     1.0
dtype: float64
```

serce pandas - DataSeries (inspirowane na języku R)

In [78]:
```
np.random.seed(55)
df = pd.DataFrame(np.random.randn(5,5),index=['A', 'B',
'C', 'D', 'E'],columns=['U', 'W', 'X', 'Y', 'Z'])
df
```

Out[78]:

	U	W	X	Y	Z
A	-1.623731	-0.101784	-1.809791	0.262654	0.259953
B	-0.381086	-0.002290	0.341615	0.897572	-0.361100
C	1.656445	-1.189009	1.666429	-2.003439	-0.477873
D	1.368799	0.258169	0.702352	0.888382	0.722220
E	-1.382103	-0.993455	-0.169466	0.287980	0.297058

In [79]:
```
df['U']
```

Out[79]:
```
A   -1.623731
B   -0.381086
C    1.656445
D    1.368799
E   -1.382103
Name: U, dtype: float64
```

In [80]:
```
df[['U', 'X']]
```

Out[80]:

	U	X
A	-1.623731	-1.809791
B	-0.381086	0.341615
C	1.656445	1.666429
D	1.368799	0.702352
E	-1.382103	-0.169466

In [81]:
```
# nowa kolumna
df['U+X'] = df['U'] + df['X']
df
```

Out[81]:

	U	W	X	Y	Z	U+X
A	-1.623731	-0.101784	-1.809791	0.262654	0.259953	-3.433522
B	-0.381086	-0.002290	0.341615	0.897572	-0.361100	-0.039471
C	1.656445	-1.189009	1.666429	-2.003439	-0.477873	3.322874
D	1.368799	0.258169	0.702352	0.888382	0.722220	2.071150
E	-1.382103	-0.993455	-0.169466	0.287980	0.297058	-1.551569

In [82]:
```
#usuwanie kolumn
df.drop('U+X',axis=1)
```

Out[82]:

	U	W	X	Y	Z
A	-1.623731	-0.101784	-1.809791	0.262654	0.259953
B	-0.381086	-0.002290	0.341615	0.897572	-0.361100
C	1.656445	-1.189009	1.666429	-2.003439	-0.477873
D	1.368799	0.258169	0.702352	0.888382	0.722220
E	-1.382103	-0.993455	-0.169466	0.287980	0.297058

In [83]:
```
df
```

Out[83]:

	U	W	X	Y	Z	U+X
A	-1.623731	-0.101784	-1.809791	0.262654	0.259953	-3.433522
B	-0.381086	-0.002290	0.341615	0.897572	-0.361100	-0.039471
C	1.656445	-1.189009	1.666429	-2.003439	-0.477873	3.322874
D	1.368799	0.258169	0.702352	0.888382	0.722220	2.071150
E	-1.382103	-0.993455	-0.169466	0.287980	0.297058	-1.551569

In [84]:
```
# zeby zmiana zaszla "w miejscu" konieczne dodanie inpl
ace=True
df.drop('U+X',axis=1, inplace=True)
```

In [85]:
```
df
```

Out[85]:

	U	W	X	Y	Z
A	-1.623731	-0.101784	-1.809791	0.262654	0.259953
B	-0.381086	-0.002290	0.341615	0.897572	-0.361100
C	1.656445	-1.189009	1.666429	-2.003439	-0.477873
D	1.368799	0.258169	0.702352	0.888382	0.722220
E	-1.382103	-0.993455	-0.169466	0.287980	0.297058

In [86]:
```
df.drop('E',axis=0)
```

Out[86]:

	U	W	X	Y	Z
A	-1.623731	-0.101784	-1.809791	0.262654	0.259953
B	-0.381086	-0.002290	0.341615	0.897572	-0.361100
C	1.656445	-1.189009	1.666429	-2.003439	-0.477873
D	1.368799	0.258169	0.702352	0.888382	0.722220

In [87]:
```
#selecting rows
df.loc['A']
```

Out[87]:
```
U   -1.623731
W   -0.101784
X   -1.809791
Y    0.262654
Z    0.259953
Name: A, dtype: float64
```

In [88]:
```
df.iloc[0]
```

Out[88]:
```
U   -1.623731
W   -0.101784
X   -1.809791
Y    0.262654
Z    0.259953
Name: A, dtype: float64
```

In [89]:
```
# w loc - najpierw row potem column
df.loc['B','Y']
```

Out[89]:
```
0.89757224571213767
```

In [90]:
```
df.loc[['A','B'],['W','Y']]
```

Out[90]:

	W	Y
A	-0.101784	0.262654
B	-0.002290	0.897572

In [91]:
```
#warunki podobnie jak w numpy
df < 1
```

Out[91]:

	U	W	X	Y	Z
A	True	True	True	True	True
B	True	True	True	True	True
C	False	True	False	True	True
D	False	True	True	True	True
E	True	True	True	True	True

In [92]:
```
# dla wartości False -> NaN
df[df < 1]
```

Out[92]:

	U	W	X	Y	Z
A	-1.623731	-0.101784	-1.809791	0.262654	0.259953
B	-0.381086	-0.002290	0.341615	0.897572	-0.361100
C	NaN	-1.189009	NaN	-2.003439	-0.477873
D	NaN	0.258169	0.702352	0.888382	0.722220
E	-1.382103	-0.993455	-0.169466	0.287980	0.297058

In [93]:
```
#wybiera rzedy takie, że..
df[df['W']>-0.5]
```

Out[93]:

	U	W	X	Y	Z
A	-1.623731	-0.101784	-1.809791	0.262654	0.259953
B	-0.381086	-0.002290	0.341615	0.897572	-0.361100
D	1.368799	0.258169	0.702352	0.888382	0.722220

In [94]:
```
#podwojne warunki
df[(df['W']>-0.5) & (df['Y'] > 0.5)]
```

Out[94]:

	U	W	X	Y	Z
B	-0.381086	-0.002290	0.341615	0.897572	-0.36110
D	1.368799	0.258169	0.702352	0.888382	0.72222

In [95]:
```
#index do domyslnych wartosci
df.reset_index()
```

Out[95]:

	index	U	W	X	Y	Z
0	A	-1.623731	-0.101784	-1.809791	0.262654	0.259953
1	B	-0.381086	-0.002290	0.341615	0.897572	-0.361100
2	C	1.656445	-1.189009	1.666429	-2.003439	-0.477873
3	D	1.368799	0.258169	0.702352	0.888382	0.722220
4	E	-1.382103	-0.993455	-0.169466	0.287980	0.297058

In [96]:
```
df['new'] = ['A_1', 'A_2', 'A_3', 'A_4', 'A_5']
df.set_index('new', inplace=True)
df
```

Out[96]:

	U	W	X	Y	Z
new					
A_1	-1.623731	-0.101784	-1.809791	0.262654	0.259953
A_2	-0.381086	-0.002290	0.341615	0.897572	-0.361100
A_3	1.656445	-1.189009	1.666429	-2.003439	-0.477873
A_4	1.368799	0.258169	0.702352	0.888382	0.722220
A_5	-1.382103	-0.993455	-0.169466	0.287980	0.297058

In [98]:
```
#multiindex
outside = ['A1','A1','A1','A2','A2','A2']
inside = [1,2,3,1,2,3]
index = list(zip(outside,inside))
print(index)
index = pd.MultiIndex.from_tuples(index)
df = pd.DataFrame(np.random.randn(6,3),index=index,colu
mns=['A','B', 'C'])
df
```

```
[('A1', 1), ('A1', 2), ('A1', 3), ('A2', 1), ('A2', 2),
('A2', 3)]
```

Out[98]:

		A	B	C
A1	**1**	0.445477	0.628428	-0.688902
	2	0.149899	1.264266	-1.214419
	3	-0.521811	-0.567562	-1.004690
A2	**1**	0.197138	0.726756	0.275635
	2	0.141798	1.552575	-0.728043
	3	-0.087340	1.298419	0.678628

In [99]:
```
df.loc['A1']
```

Out[99]:

	A	B	C
1	0.445477	0.628428	-0.688902
2	0.149899	1.264266	-1.214419
3	-0.521811	-0.567562	-1.004690

In [100]:
```
df.loc['A1'].loc[1]
```

Out[100]:
```
A    0.445477
B    0.628428
C   -0.688902
Name: 1, dtype: float64
```

In [101]:
```
df.index.names = ['Group','Number']
df
```

Out[101]:

		A	B	C
Group	**Number**			
A1	**1**	0.445477	0.628428	-0.688902
	2	0.149899	1.264266	-1.214419
	3	-0.521811	-0.567562	-1.004690
A2	**1**	0.197138	0.726756	0.275635
	2	0.141798	1.552575	-0.728043
	3	-0.087340	1.298419	0.678628

In [102]:
```
df.xs('A1')
```

Out[102]:

	A	B	C
Number			
1	0.445477	0.628428	-0.688902
2	0.149899	1.264266	-1.214419
3	-0.521811	-0.567562	-1.004690

In [103]:
```
df.xs(1,level='Number')
```

Out[103]:

	A	B	C
Group			
A1	0.445477	0.628428	-0.688902
A2	0.197138	0.726756	0.275635

In [104]:
```
#brak danych
df = pd.DataFrame({'A':[1,2,np.nan],
                   'B':[5,np.nan,np.nan],
                   'C':[1,2,3]})
df
```

Out[104]:

	A	B	C
0	1.0	5.0	1
1	2.0	NaN	2
2	NaN	NaN	3

In [105]:
```
df.dropna()
```

Out[105]:

	A	B	C
0	1.0	5.0	1

In [106]:
```
df.dropna(axis=1)
```

Out[106]:

	C
0	1
1	2
2	3

In [107]:
```
df.dropna(thresh=2)
```

Out[107]:

	A	B	C
0	1.0	5.0	1
1	2.0	NaN	2

In [108]:
```
#uzupelnianie brakujacych danych
df.fillna(value='FILL VALUE')
```

Out[108]:

	A	B	C
0	1	5	1
1	2	FILL VALUE	2
2	FILL VALUE	FILL VALUE	3

In [109]:
```
#Groupby
data = {'Firma':['McDonalds', 'McDonalds','KFC','KFC','
Starbucks','Starbucks'],
       'Osoba':['Ania','Kasia','Piotr','Mateusz','Basia
','Szymon'],
       'Sprzedaż':[400,250,222,333,200,120]}
df = pd.DataFrame(data)
df
```

Out[109]:

	Firma	Osoba	Sprzedaż
0	McDonalds	Ania	400
1	McDonalds	Kasia	250
2	KFC	Piotr	222
3	KFC	Mateusz	333
4	Starbucks	Basia	200
5	Starbucks	Szymon	120

In [110]:
```
by_firma = df.groupby('Firma')
```

In [111]:
```
by_firma.mean()
```

Out[111]:

	Sprzedaż
Firma	
KFC	277.5
McDonalds	325.0
Starbucks	160.0

In [112]:
```
df.groupby('Firma').mean()
```

Out[112]:

	Sprzedaż
Firma	
KFC	277.5
McDonalds	325.0
Starbucks	160.0

In [113]:
```
by_firma.max()
```

Out[113]:

	Osoba	Sprzedaż
Firma		
KFC	Piotr	333
McDonalds	Kasia	400
Starbucks	Szymon	200

In [114]:
```
by_firma.describe().transpose()
```

Out[114]:

Firma	KFC								McD
	count	mean	std	min	25%	50%	75%	max	cou
Sprzedaż	2.0	277.5	78.488853	222.0	249.75	277.5	305.25	333.0	2.0

1 rows × 24 columns

In [115]:
```
# laczenie DataSeries
df1 = pd.DataFrame({'A': [1,2,3,4],
                        'B': [5,6,7,8],
                        'C': [9,10,11,12],},
                        index=[0, 1, 2, 3])

df2 = pd.DataFrame({'A': [11,12,13,14],
                        'B': [15,16,17,18],
                        'C': [19,20,21,22],},
                        index=[0, 1, 2, 3])
```

In [116]:
```
df1
```

Out[116]:

	A	B	C
0	1	5	9
1	2	6	10
2	3	7	11
3	4	8	12

In [117]:
```
df2
```

Out[117]:

	A	B	C
0	11	15	19
1	12	16	20
2	13	17	21
3	14	18	22

In [118]:
```
pd.concat([df1,df2])
```

Out[118]:

	A	B	C
0	1	5	9
1	2	6	10
2	3	7	11
3	4	8	12
0	11	15	19
1	12	16	20
2	13	17	21
3	14	18	22

In [119]:
```
df3 = pd.concat([df1,df2], axis=1)
df3
```

Out[119]:

	A	B	C	A	B	C
0	1	5	9	11	15	19
1	2	6	10	12	16	20
2	3	7	11	13	17	21
3	4	8	12	14	18	22

In [121]:
```
#dwie kolumny o tej samej nazwie...
df3['A']
```

Out[121]:

	A	A
0	1	11
1	2	12
2	3	13
3	4	14

In [122]:
```
df_left = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3']
,
                        'A': ['A0', 'A1', 'A2', 'A3'],
                        'B': ['B0', 'B1', 'B2', 'B3']})

df_right = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3'
],
                             'C': ['C0', 'C1', 'C2', 'C3']
,
                             'D': ['D0', 'D1', 'D2', 'D3']
})
```

In [123]:
```
df_left
```

Out[123]:

	A	B	key
0	A0	B0	K0
1	A1	B1	K1
2	A2	B2	K2
3	A3	B3	K3

In [124]:
```
df_right
```

Out[124]:

	C	D	key
0	C0	D0	K0
1	C1	D1	K1
2	C2	D2	K2
3	C3	D3	K3

In [125]:
```
#merge
pd.merge(df_left,df_right,how='inner',on='key')
```

Out[125]:

	A	B	key	C	D
0	A0	B0	K0	C0	D0
1	A1	B1	K1	C1	D1
2	A2	B2	K2	C2	D2
3	A3	B3	K3	C3	D3

In [126]:
```
#join
df_left.join(df_right)
```

```
---------------------------------------------------------------
------------------
ValueError                                Traceback (mos
t recent call last)
<ipython-input-126-2021e80741d5> in <module>()
      1 #join
----> 2 df_left.join(df_right)

/usr/local/lib/python3.5/dist-packages/pandas/core/frame
.py in join(self, other, on, how, lsuffix, rsuffix, sort
)
   4553         # For SparseDataFrame's benefit
   4554         return self._join_compat(other, on=on, h
ow=how, lsuffix=lsuffix,
-> 4555                                  rsuffix=rsuffix
, sort=sort)
   4556
   4557     def _join_compat(self, other, on=None, how='
left', lsuffix='', rsuffix='',

/usr/local/lib/python3.5/dist-packages/pandas/core/frame
.py in _join_compat(self, other, on, how, lsuffix, rsuff
ix, sort)
   4567             return merge(self, other, left_on=on
, how=how,
   4568                          left_index=on is None,
right_index=True,
-> 4569                          suffixes=(lsuffix, rsuf
fix), sort=sort)
   4570         else:
   4571             if on is not None:

/usr/local/lib/python3.5/dist-packages/pandas/tools/merg
e.py in merge(left, right, how, on, left_on, right_on, l
eft_index, right_index, sort, suffixes, copy, indicator)
     60                          right_index=right_index
, sort=sort, suffixes=suffixes,
     61                          copy=copy, indicator=in
dicator)
---> 62     return op.get_result()
     63 if __debug__:
     64     merge.__doc__ = _merge_doc % '\nleft : DataF
rame'

/usr/local/lib/python3.5/dist-packages/pandas/tools/merg
e.py in get_result(self)
    554
    555         llabels, rlabels = items_overlap_with_su
ffix(ldata.items, lsuf,
--> 556
rdata.items, rsuf)
    557
    558         lindexers = {1: left_indexer} if left_in
dexer is not None else {}

/usr/local/lib/python3.5/dist-packages/pandas/core/inter
```

In [127]:
```
df_left.join(df_right, rsuffix='right')
```

Out[127]:

	A	B	key	C	D	keyright
0	A0	B0	K0	C0	D0	K0
1	A1	B1	K1	C1	D1	K1
2	A2	B2	K2	C2	D2	K2
3	A3	B3	K3	C3	D3	K3

In [128]:
```
# operacje na DataFrame
df = pd.DataFrame({'col1':[1,2,3,4],'col2':[444,555,666
,444],'col3':['abc','def','ghi','xyz']})
#wyswietlanie fragmentu DataFrame
df.head()
```

Out[128]:

	col1	col2	col3
0	1	444	abc
1	2	555	def
2	3	666	ghi
3	4	444	xyz

In [129]:
```
df['col2'].unique()
```

Out[129]:
```
array([444, 555, 666])
```

In [130]:
```
df['col2'].value_counts()
```

Out[130]:
```
444    2
555    1
666    1
Name: col2, dtype: int64
```

In [131]:
```
# warunkowe ograniczanie danych
new_df = df[(df['col1']>2) & (df['col2']==444)]
new_df
```

Out[131]:

	col1	col2	col3
3	4	444	xyz

In [132]:
```
df['col4'] = df['col3'].apply(len)
df
```

Out[132]:

	col1	col2	col3	col4
0	1	444	abc	3
1	2	555	def	3
2	3	666	ghi	3
3	4	444	xyz	3

In [133]:
```
df['col1'].sum()
```

Out[133]:
```
10
```

In [134]:
```
#usuwanie kolumn
del df['col1']
df
```

Out[134]:

	col2	col3	col4
0	444	abc	3
1	555	def	3
2	666	ghi	3
3	444	xyz	3

In [135]:
```
#wartosci index i columns
df.columns
```

Out[135]:
```
Index(['col2', 'col3', 'col4'], dtype='object')
```

In [136]:
```
df.index
```

Out[136]:
```
RangeIndex(start=0, stop=4, step=1)
```

In [137]:
```
df.sort_values(by='col2') # inplace!
```

Out[137]:

	col2	col3	col4
0	444	abc	3
3	444	xyz	3
1	555	def	3
2	666	ghi	3

In [138]:
```
#znajdowanie wartosci pustych
df.isnull()
```

Out[138]:

	col2	col3	col4
0	False	False	False
1	False	False	False
2	False	False	False
3	False	False	False

In [139]:
```
# input and output
df = pd.read_csv('example')
df
```

Out[139]:

	a	b	c	d
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15

In [140]:
```
df.to_csv('example',index=False)
```

In [141]:
```
pd.read_excel('Excel_Sample.xlsx',sheetname='Sheet1')
```

Out[141]:

	a	b	c	d
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11
3	12	13	14	15

In [142]:
```
df.to_excel('Excel_Sample.xlsx',sheet_name='Sheet1')
```

ćwiczenie 2

zaimportuj dane z pliku

https://mdcune.psych.ucla.edu/modules/bioinformatics/extras/QTL_Sample_data.xls/view (https://mdcune.psych.ucla.edu/modules/bioinformatics/extras/QTL_Sample_data.xls/view)

sprawdź strukturę tabeli

sprawdź jaki jest średni wiek (age)

sprawdź jaka jest najwyższa wartość brainwt

sprawdź jaka jest płeć (sex) myszy o ID 1709

sprawdź jakie ID ma mysz o najwyższej wartości bodywt

sprawdź ile jest myszy płci męskiej i żeńskiej

Sprawdź ile jest rodzajów linii myszy (Strain)

Sprawdź jaka jest średnia wartość brainwt i bodywt odpowiednio dla myszy żeńskich i męskich

In [143]:
```
# Literatura/kursy:
# Python Data Science Handbook - O'Reilly Media
# udemy - Python Data Science and Machine Learning Boot camp
# coursera - Introduction to Data Science in Python by University of Michigan
# datacamp - https://www.datacamp.com/courses/intro-to-python-for-data-science
# edX - https://www.edx.org/course/introduction-python-data-science-microsoft-dat208x-5
# stackoverflow
# http://pandas.pydata.org/pandas-docs/stable/tutorials.html
# Python vs R article - http://www.kdnuggets.com/2015/05/r-vs-python-data-science.html
```