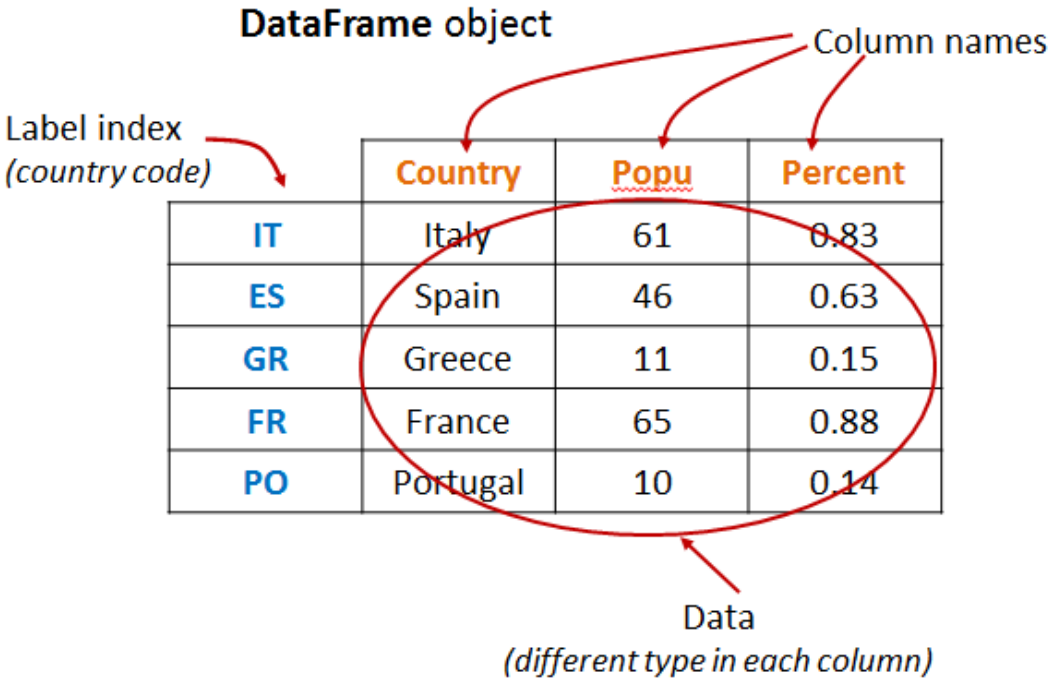
Основы **Pandas**

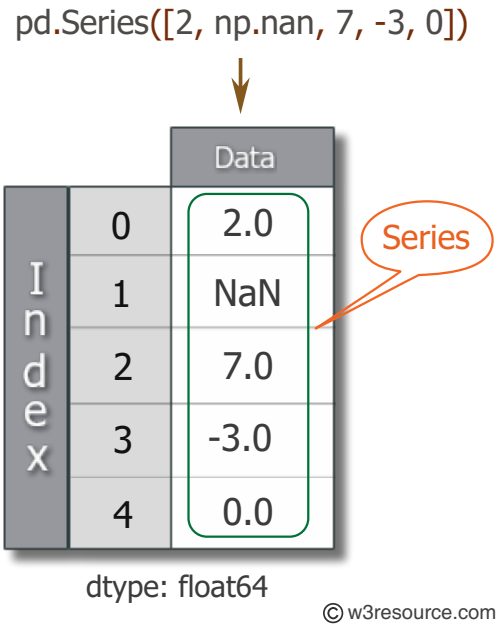
Основные объекты и понятия

Основу **Pandas** составляют такие структуры данных, как **Series и DataFrame**.

Series - это **одномерный массив,** имеющий специальные **метки (индексы)** и способные хранить даннеы **любого** типа.

DataFrame - это **двумерный массив (матрица, таблица)**, имеющий **специальные метки (индексы),** который хранит в своих столбцах данные **разных типов.**





In [1]:

```
import numpy as np
```

```
import pandas as pd
```

Series

In [7]:

```
data = ["Pandas", "Matplotlib", "Numpy"]
s = pd.Series(data)
s
```

Out[7]:

```
0         Pandas
1     Matplotlib
2          Numpy
dtype: object
```

In [8]:

```
s = pd.Series([2, np.nan, 7, -3, 0])
s
```

Out[8]:

```
0    2.0
1    NaN
2    7.0
3   -3.0
4    0.0
dtype: float64
```

In [9]:

```
s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd'])
s
```

Out[9]:

```
a    3
b   -5
c    7
d    4
dtype: int64
```

In [10]:

```
s = pd.Series(np.random.randn(6), index=['p', 'q', 'r', 'n', 't','v'])
s
```

Out[10]:

```
p    0.447376
q   -0.377268
r   -2.185190
n    1.312263
t    1.333090
v    0.888254
dtype: float64
```

In [11]:

```
s.index
```

Out[11]:

```
Index(['p', 'q', 'r', 'n', 't', 'v'], dtype='object')
```

In [12]:

```
n = {'q': 1, 'p': 2, 'r': 3}
pd.Series(n)
```

Out[12]:

```
q    1
p    2
r    3
dtype: int64
```

In [17]:

```
print(s.dtype)
print(s.shape)
print(s.ndim)
print(s.size)
```

```
float64
(6,)
1
6
```

In [19]:

```
s.to_numpy()
```

Out[19]:

```
array([ 0.44737622, -0.37726814, -2.18519004,  1.31226324,  1.3330903 ,
        0.88825438])
```

Индексы и выборка

In [36]:

```
print(s, '\n')


print(s[0], '\n')


print(s[1:5], '\n')


print(s[s > 0], '\n')


print(s[(s > 0) & (s < 1)], '\n')


print(s['t'], '\n')

print(s[['r','t']])
```

```
p    0.447376
q   -0.377268
r   -2.185190
n    1.312263
t    1.333090
v    0.888254
dtype: float64

0.4473762234924483

q   -0.377268
r   -2.185190
n    1.312263
t    1.333090
dtype: float64

p    0.447376
n    1.312263
t    1.333090
v    0.888254
```

```
dtype: float64

p    0.447376
v    0.888254
dtype: float64

1.3330903003322911

r   -2.18519
t    1.33309
dtype: float64
```

In [29]:

```
s[[4,2]]
```

Out[29]:

```
t    1.33309
r   -2.18519
dtype: float64
```

In [33]:

```
print(np.sin(s), '\n')
print(np.abs(s), '\n')
print(np.tan(s), '\n')
```

```
p    0.432601
q   -0.368382
r   -0.817123
n    0.966766
t    0.971881
v    0.775972
dtype: float64

p    0.447376
q    0.377268
r    2.185190
n    1.312263
t    1.333090
v    0.888254
dtype: float64

p    0.479823
q   -0.396248
r    1.417477
n    3.781413
t    4.127342
v    1.230203
dtype: float64
```

Добавление и удаление

In [38]:

```
s['pylounge'] = 1000
s
```

Out[38]:

```
p              0.447376
q             -0.377268
r             -2.185190
n              1.312263
t              1.333090
v              0.888254
pylounge    1000.000000
dtype: float64
```

In [40]:

```
s.drop(labels=['p', 'pylounge']) # labels, index, axis и т. д.
```

Out[40]:

```
q   -0.377268
r   -2.185190
n    1.312263
t    1.333090
v    0.888254
dtype: float64
```

In [41]:

```
print(s.sum(), '\n')
print(s.mean(), '\n')
print(s.max())
```

```
1001.4185259581628

143.0597894225947

1000.0
```

Операции

In [42]:

```
s + s
```

Out[42]:

```
p              0.894752
q             -0.754536
r             -4.370380
n              2.624526
t              2.666181
v              1.776509
pylounge    2000.000000
dtype: float64
```

In [43]:

```
s * s
```

Out[43]:

```
p                  0.200145
q                  0.142331
r                  4.775056
n                  1.722035
t                  1.777130
v                  0.788996
pylounge    1000000.000000
dtype: float64
```

In [44]:

```
s ** 2
```

Out[44]:

```
p                  0.200145
q                  0.142331
r                  4.775056
n                  1.722035
t                  1.777130
v                  0.788996
pylounge    1000000.000000
dtype: float64
```

In [45]:

```
s.astype(np.int16)
```

Out[45]:

```
p              0
q              0
r             -2
n              1
t              1
v              0
pylounge    1000
dtype: int16
```

In [54]:

```
pd.read_csv('./heart.csv', nrows=1)
```

Out[54]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	1	145	233	1	2	150	0	2.3	3	0	fixed	0

In [57]:

```
s.to_csv('./myDataFrame.csv')

#pd.read_excel('file.xlsx')
#pd.to_excel('dir/myDataFrame.xlsx', sheet_name='Sheet1')
#df.to_json(filename)

#xlsx = pd.ExcelFile('file.xls')
#df = pd.read_excel(xlsx, 'Sheet1')
#pd.read_json(json_string)
#pd.read_html(url)
#pd.read_clipboard()
```

In [58]:

```
'''from sqlalchemy import create_engine

engine = create_engine('sqlite:///:memory:')
pd.read_sql("SELECT * FROM my_table;", engine)
pd.read_sql_table('my_table', engine)
pd.read_sql_query("SELECT * FROM my_table;", engine)

pd.to_sql('myDf', engine)'''
```

Out[58]:

```
'from sqlalchemy import create_engine\n\nengine = create_engine(\'sqlite:///:memory:\')\n
pd.read_sql("SELECT * FROM my_table;", engine)\npd.read_sql_table(\'my_table\', engine)\n
pd.read_sql_query("SELECT * FROM my_table;", engine)\n\npd.to_sql(\'myDf\', engine)'
```

DataFrame

In [8]:

```
data = [[4, 7, 10],[5, 8, 11],[6, 9, 12]]
df= pd.DataFrame(data)
df
```

Out[8]:

	0	1	2
0	4	7	10
1	5	8	11

	0	1	2
2	6	9	12

In [11]:

```
df = pd.DataFrame(data, index=['a', 'b', 'c'], columns=['X', 'Y', 'Z'])
df
```

Out[11]:

	X	Y	Z
a	4	7	10
b	5	8	11
c	6	9	12

In [12]:

```
data = {'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],
        'Age':[27, 24, 22, 32],
        'Address':['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],
        'Qualification':['Msc', 'MA', 'MCA', 'Phd']}

df = pd.DataFrame(data)
df
```

Out[12]:

	Name	Age	Address	Qualification
0	Jai	27	Delhi	Msc
1	Princi	24	Kanpur	MA
2	Gaurav	22	Allahabad	MCA
3	Anuj	32	Kannauj	Phd

In [48]:

```
df = pd.read_csv('./heart.csv',  sep=',', header = None, nrows=10)
df
```

Out[48]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
1	63	1	1	145	233	1	2	150	0	2.3	3	0	fixed	0
2	67	1	4	160	286	0	2	108	1	1.5	2	3	normal	1
3	67	1	4	120	229	0	2	129	1	2.6	2	2	reversible	0
4	37	1	3	130	250	0	0	187	0	3.5	3	0	normal	0
5	41	0	2	130	204	0	2	172	0	1.4	1	0	normal	0
6	56	1	2	120	236	0	0	178	0	0.8	1	0	normal	0
7	62	0	4	140	268	0	2	160	0	3.6	3	2	normal	1
8	57	0	4	120	354	0	0	163	1	0.6	1	0	normal	0
9	63	1	4	130	254	0	2	147	0	1.4	2	1	reversible	1

In [22]:

```
df.head(5)
```

Out[22]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
1	63	1	1	145	233	1	2	150	0	2.3	3	0	fixed	0
2	67	1	4	160	286	0	2	108	1	1.5	2	3	normal	1
3	67	1	4	120	229	0	2	129	1	2.6	2	2	reversible	0
4	37	1	3	130	250	0	0	187	0	3.5	3	0	normal	0

In [23]:

```
df.tail(5)
```

Out[23]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
5	41	0	2	130	204	0	2	172	0	1.4	1	0	normal	0
6	56	1	2	120	236	0	0	178	0	0.8	1	0	normal	0
7	62	0	4	140	268	0	2	160	0	3.6	3	2	normal	1
8	57	0	4	120	354	0	0	163	1	0.6	1	0	normal	0
9	63	1	4	130	254	0	2	147	0	1.4	2	1	reversible	1

In [24]:

```
df.info() #  Index, Datatype and Memory information
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 14 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   0       10 non-null     object
 1   1       10 non-null     object
 2   2       10 non-null     object
 3   3       10 non-null     object
 4   4       10 non-null     object
 5   5       10 non-null     object
 6   6       10 non-null     object
 7   7       10 non-null     object
 8   8       10 non-null     object
 9   9       10 non-null     object
 10  10      10 non-null     object
 11  11      10 non-null     object
 12  12      10 non-null     object
 13  13      10 non-null     object
dtypes: object(14)
memory usage: 1.2+ KB
```

In [26]:

```
df.describe() #Summary statistics for numerical columns
```

Out[26]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
count	10	10	10	10	10	10	10	10	10	10	10	10	10	10
unique	8	3	5	6	10	3	3	10	3	9	4	5	4	3
top	63	1	4	120	354	0	2	187	0	1.4	1	0	normal	0
freq	2	6	5	3	1	8	6	1	6	2	3	5	6	6

In [51]:

```
print(df.shape, '\n')
```

```
print(df.ndim, '\n')
print(df.size, '\n')
print(df.index,'  ', df.columns, '\n')
print(df.count()) #Number of non-NA values
```

```
(10, 14)

2

140

RangeIndex(start=0, stop=10, step=1)    Int64Index([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
12, 13], dtype='int64')

0     10
1     10
2     10
3     10
4     10
5     10
6     10
7     10
8     10
9     10
10    10
11    10
12    10
13    10
dtype: int64
```

In [27]:

```
df.value_counts()
```

Out[27]:

```
0    1    2    3         4     5    6        7        8      9        10     11  12
13
age  sex  cp   trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  ca  thal
target    1
67   1    4    160       286   0    2        108      1      1.5      2      3   normal
1         1
               120       229   0    2        129      1      2.6      2      2   reversib
le  0         1
63   1    4    130       254   0    2        147      0      1.4      2      1   reversib
le  1         1
          1    145       233   1    2        150      0      2.3      3      0   fixed
0         1
62   0    4    140       268   0    2        160      0      3.6      3      2   normal
1         1
57   0    4    120       354   0    0        163      1      0.6      1      0   normal
0         1
56   1    2    120       236   0    0        178      0      0.8      1      0   normal
0         1
41   0    2    130       204   0    2        172      0      1.4      1      0   normal
0         1
37   1    3    130       250   0    0        187      0      3.5      3      0   normal
0         1
dtype: int64
```

In [58]:

```
df = pd.read_csv('./heart.csv',  sep=',', nrows=10)
df["sex"].value_counts()
```

Out[58]:

```
1    7
0    3
Name: sex, dtype: int64
```

In [57]:

```
df.index = pd.date_range('1900/1/30', periods=df.shape[0])
df
```

Out[57]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
1900-01-30	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
1900-01-31	63	1	1	145	233	1	2	150	0	2.3	3	0	fixed	0
1900-02-01	67	1	4	160	286	0	2	108	1	1.5	2	3	normal	1
1900-02-02	67	1	4	120	229	0	2	129	1	2.6	2	2	reversible	0
1900-02-03	37	1	3	130	250	0	0	187	0	3.5	3	0	normal	0
1900-02-04	41	0	2	130	204	0	2	172	0	1.4	1	0	normal	0
1900-02-05	56	1	2	120	236	0	0	178	0	0.8	1	0	normal	0
1900-02-06	62	0	4	140	268	0	2	160	0	3.6	3	2	normal	1
1900-02-07	57	0	4	120	354	0	0	163	1	0.6	1	0	normal	0
1900-02-08	63	1	4	130	254	0	2	147	0	1.4	2	1	reversible	1

In [60]:

```
df["sex"].nunique() # сколько значений
```

Out[60]:

```
2
```

In [61]:

```
df["sex"].unique() # уникальные значения
```

Out[61]:

```
array([1, 0])
```

In [70]:

```
df = pd.DataFrame({
'country': ['Kazakhstan', 'Russia', 'Belarus', 'Ukraine','Kazakhstan'],
'population': [17.04, 143.5, 9.5, 45.5, 232.12],
'square': [2724902, 17125191, 207600, 603628, 35445]
}, index=['KZ', 'RU', 'BY', 'UA', 'KZ'])
df
```

Out[70]:

	country	population	square
KZ	Kazakhstan	17.04	2724902
RU	Russia	143.50	17125191
BY	Belarus	9.50	207600
UA	Ukraine	45.50	603628
KZ	Kazakhstan	232.12	35445

In [73]:

```
print(df["country"].nunique(), ' ', df["country"].unique())
```

```
4   ['Kazakhstan' 'Russia' 'Belarus' 'Ukraine']
```

In [72]:

```
df["country"].value_counts()
```

Out[72]:

```
Kazakhstan    2
Belarus       1
Russia        1
Ukraine       1
Name: country, dtype: int64
```

In [213]:

```
df['Name'].apply(lambda x: x.upper()) # применяет функцию для выбранных столбцов или всех
```

Out[213]:

```
0                        ALLEN, MISS ELISABETH WALTON
1                         ALLISON, MISS HELEN LORAINE
2                  ALLISON, MR HUDSON JOSHUA CREIGHTON
3        ALLISON, MRS HUDSON JC (BESSIE WALDO DANIELS)
4                       ALLISON, MASTER HUDSON TREVOR
                              ...
1308                             ZAKARIAN, MR ARTUN
1309                        ZAKARIAN, MR MAPRIEDER
1310                              ZENNI, MR PHILIP
1311                              LIEVENS, MR RENE
1312                                ZIMMERMAN, LEO
Name: Name, Length: 1313, dtype: object
```

Индексация и фильтрация

In [74]:

```
df["country"] # df.country
```

Out[74]:

```
KZ    Kazakhstan
RU        Russia
BY       Belarus
UA       Ukraine
KZ    Kazakhstan
Name: country, dtype: object
```

In [75]:

```
df[["country", "square"]]
```

Out[75]:

	country	square
KZ	**Kazakhstan**	**2724902**
RU	**Russia**	**17125191**
BY	**Belarus**	**207600**
UA	**Ukraine**	**603628**
KZ	**Kazakhstan**	**35445**

Доступ к строкам по индексу возможен несколькими способами:

- **.loc -** используется для доступа к значению по строковой метке (индексу) **(by row & column index label)**
- **.iloc -** используется для доступа по числовому значению (начиная от **0**) **(by row & column number)**

  .at и **.iat** анаологичны, но можут получать только одно значение за раз

  ix - устарел

In [103]:

```
df.loc['KZ'] # at
```

Out[103]:

	country	population	square
KZ	Kazakhstan	17.04	2724902
KZ	Kazakhstan	232.12	35445

In [107]:

```
df.loc[['KZ'], ['square']] # строка с меткой KZ, стобец с меткой square
```

Out[107]:

	square
KZ	2724902
KZ	35445

In [106]:

```
df.iloc[1] # 1 строка, все стобцы , iat
```

Out[106]:

```
country          Russia
population        143.5
square         17125191
Name: RU, dtype: object
```

In [105]:

```
df.iloc[1, 1] # 1 строка и 1 столбец
```

Out[105]:

```
143.5
```

In [109]:

```
df.iloc[1, 1:3] # 1 строка с 1 по 3 столбец
```

Out[109]:

```
population        143.5
square         17125191
Name: RU, dtype: object
```

In [111]:

```
df.iloc[1:3, 1:3] # с 1 по 3 строки и с 1 по 3 столбец
```

Out[111]:

	population	square
RU	143.5	17125191
BY	9.5	207600

In [87]:

```
df.loc[['KZ', 'RU'], 'square'] # по индексу и интересующим колонкам
```

Out[87]:

```
KZ     2724902
KZ       35445
RU    17125191
Name: square, dtype: int64
```

In [97]:

```
df.loc['RU':'UA', :]
```

Out[97]:

	country	population	square
RU	Russia	143.5	17125191
BY	Belarus	9.5	207600
UA	Ukraine	45.5	603628

In [100]:

```
print(df[df.population > 30], '\n')

print(df[df.population > 30][['country', 'square']])
```

```
       country  population    square
RU      Russia      143.50  17125191
UA     Ukraine       45.50    603628
KZ  Kazakhstan      232.12     35445

       country    square
RU      Russia  17125191
UA     Ukraine    603628
KZ  Kazakhstan     35445
```

In [116]:

```
df['Density'] = df['population'] / df['square'] * 1000000
df
```

Out[116]:

	country	population	square	Density
KZ	Kazakhstan	17.04	2724902	6.253436
RU	Russia	143.50	17125191	8.379469
BY	Belarus	9.50	207600	45.761079
UA	Ukraine	45.50	603628	75.377550
KZ	Kazakhstan	232.12	35445	6548.737481

In [122]:

```
print(df.sum(), '\n\n', df.mean(), '\n\n',df.max())
```

```
country       KazakhstanRussiaBelarusUkraineKazakhstan
population                                       447.66
square                                         20696766
Density                                        6684.51
dtype: object

 population    8.953200e+01
square        4.139353e+06
Density       1.336902e+03
dtype: float64

 country        Ukraine
population      232.12
square        17125191
Density        6548.74
dtype: object
```

In [123]:

```
df['square'].mean()
```

Out[123]:

4139353.2

In [130]:

```
df.drop(['square'], axis=1)
```

Out[130]:

	country	population	Density
KZ	Kazakhstan	17.04	6.253436
RU	Russia	143.50	8.379469
BY	Belarus	9.50	45.761079
UA	Ukraine	45.50	75.377550
KZ	Kazakhstan	232.12	6548.737481

Обработка пустых значений

In [177]:

```
df = pd.read_csv('titanic.csv')
df.head(10)
```

Out[177]:

	PassengerID	Name	PClass	Age	Sex	Survived	SexCode
0	1	Allen, Miss Elisabeth Walton	1st	29.00	female	1	1
1	2	Allison, Miss Helen Loraine	1st	2.00	female	0	1
2	3	Allison, Mr Hudson Joshua Creighton	1st	30.00	male	0	0
3	4	Allison, Mrs Hudson JC (Bessie Waldo Daniels)	1st	25.00	female	0	1
4	5	Allison, Master Hudson Trevor	1st	0.92	male	1	0
5	6	Anderson, Mr Harry	1st	47.00	male	1	0
6	7	Andrews, Miss Kornelia Theodosia	1st	63.00	female	1	1
7	8	Andrews, Mr Thomas, jr	1st	39.00	male	0	0
8	9	Appleton, Mrs Edward Dale (Charlotte Lamson)	1st	58.00	female	1	1
9	10	Artagaveytia, Mr Ramon	1st	71.00	male	0	0

Для определения **NaN**значений панды использует либо **.isna()**или **.isnull().** Эти **NaN**значения унаследованы от того **,** что панды построены на вершине **NumPy,** в то время как имена двух функций происходят из **DataFrames** АиРа**,** чья структура и функциональность панд пытались имитировать**.**

In [178]:

```
df.isna()[1000:]
```

Out[178]:

	PassengerID	Name	PClass	Age	Sex	Survived	SexCode
1000	False	False	False	True	False	False	False
1001	False	False	False	True	False	False	False
1002	False	False	False	True	False	False	False
1003	False	False	False	True	False	False	False
1004	False	False	False	True	False	False	False

	PassengerID	Name	PClass	Age	Sex	Survived	SexCode
~~1308~~	~~False~~	~~False~~	~~False~~	~~False~~	~~False~~	~~False~~	~~False~~
1309	False	False	False	False	False	False	False
1310	False	False	False	False	False	False	False
1311	False	False	False	False	False	False	False
1312	False	False	False	False	False	False	False

313 rows × 7 columns

In [179]:

```
df.iloc[1000]
```

Out[179]:

```
PassengerID                  1001
Name            McCoy, Miss Agnes
PClass                        3rd
Age                           NaN
Sex                        female
Survived                        0
SexCode                         1
Name: 1000, dtype: object
```

In [180]:

```
len(df)
```

Out[180]:

```
1313
```

In [181]:

```
df2 = df.dropna() # axis=1 выкидываем все стобцы с  null values, axis=0 - строки
len(df2)
```

Out[181]:

```
756
```

In [182]:

```
df = df.fillna(df['Age'].mean())
df.iloc[1000]
```

Out[182]:

```
PassengerID                  1001
Name            McCoy, Miss Agnes
PClass                        3rd
Age                        30.398
Sex                        female
Survived                        0
SexCode                         1
Name: 1000, dtype: object
```

Группировка и сортировка

In [185]:

```
df.sort_values('Age',ascending=False).head(10) #sort_index
```

Out[185]:

	PassengerID	Name	PClass	Age	Sex	Survived	SexCode
505	506	Mitchell, Mr Henry Michael	2nd	71.0	male	0	0

	PassengerID	Name	PClass	Age	Sex	Survived	SexCode
9	10	Artagaveytia, Mr Ramon	1st	71.0	male	0	0
119	120	Goldschmidt, Mr George B	1st	71.0	male	0	0
72	73	Crosby, Captain Edward Gifford	1st	70.0	male	0	0
73	74	Crosby, Mrs Edward Gifford (Catherine Elizabet...	1st	69.0	female	1	1
252	253	Straus, Mr Isidor	1st	67.0	male	0	0
772	773	Dewan, Mr Frank	3rd	65.0	male	0	0
179	180	Millet, Mr Francis Davis	1st	65.0	male	0	0
103	104	Fortune, Mr Mark	1st	64.0	male	0	0
67	68	Compton, Mrs Alexander Taylor (Mary Eliza Inge...	1st	64.0	female	1	1

In [190]:

```
df.groupby(['Sex'])['PassengerID'].count()
```

Out[190]:

```
Sex
female    462
male      851
Name: PassengerID, dtype: int64
```

In [196]:

```
df.groupby(['Sex'])['Age'].agg(np.mean)
```

Out[196]:

```
Sex
female    29.773637
male      30.736945
Name: Age, dtype: float64
```

In [198]:

```
pvt = titanic_df.pivot_table(index=['Sex'], columns=['PClass'], values='Name', aggfunc='
count')
pvt # сводная таблица сколько всего женщин и мужчин было в конкретном классе корабля
```

Out[198]:

PClass	*	1st	2nd	3rd
Sex				
female	NaN	143.0	107.0	212.0
male	1.0	179.0	172.0	499.0

Объединение и комбинации

In [215]:

```
df.append(df) # добавляет строки df1 в конец df2
```

Out[215]:

	PassengerID	Name	PClass	Age	Sex	Survived	SexCode
0	1	Allen, Miss Elisabeth Walton	1st	29.00	female	1	1
1	2	Allison, Miss Helen Loraine	1st	2.00	female	0	1
2	3	Allison, Mr Hudson Joshua Creighton	1st	30.00	male	0	0
3	4	Allison, Mrs Hudson JC (Bessie Waldo Daniels)	1st	25.00	female	0	1
4	5	Allison, Master Hudson Trevor	1st	0.92	male	1	0

...	PassengerID	Name	PClass	Age	Sex	Survived	SexCode
1308	1309	Zakarian, Mr Artun	3rd	27.00	male	0	0
1309	1310	Zakarian, Mr Maprieder	3rd	26.00	male	0	0
1310	1311	Zenni, Mr Philip	3rd	22.00	male	0	0
1311	1312	Lievens, Mr Rene	3rd	24.00	male	0	0
1312	1313	Zimmerman, Leo	3rd	29.00	male	0	0

2626 rows × 7 columns

In [216]:

```
pd.concat([df, df],axis=1) # добавляет столбцы  df1 в конец df2
```

Out[216]:

	PassengerID	Name	PClass	Age	Sex	Survived	SexCode	PassengerID	Name	PClass	Age	Sex
0	1	Allen, Miss Elisabeth Walton	1st	29.00	female	1	1	1	Allen, Miss Elisabeth Walton	1st	29.00	female
1	2	Allison, Miss Helen Loraine	1st	2.00	female	0	1	2	Allison, Miss Helen Loraine	1st	2.00	female
2	3	Allison, Mr Hudson Joshua Creighton	1st	30.00	male	0	0	3	Allison, Mr Hudson Joshua Creighton	1st	30.00	male
3	4	Allison, Mrs Hudson JC (Bessie Waldo Daniels)	1st	25.00	female	0	1	4	Allison, Mrs Hudson JC (Bessie Waldo Daniels)	1st	25.00	female
4	5	Allison, Master Hudson Trevor	1st	0.92	male	1	0	5	Allison, Master Hudson Trevor	1st	0.92	male
...	...	...	...	...	...	...	...	...	...	...	...	...
1308	1309	Zakarian, Mr Artun	3rd	27.00	male	0	0	1309	Zakarian, Mr Artun	3rd	27.00	male
1309	1310	Zakarian, Mr Maprieder	3rd	26.00	male	0	0	1310	Zakarian, Mr Maprieder	3rd	26.00	male
1310	1311	Zenni, Mr Philip	3rd	22.00	male	0	0	1311	Zenni, Mr Philip	3rd	22.00	male
1311	1312	Lievens, Mr Rene	3rd	24.00	male	0	0	1312	Lievens, Mr Rene	3rd	24.00	male
1312	1313	Zimmerman, Leo	3rd	29.00	male	0	0	1313	Zimmerman, Leo	3rd	29.00	male

1313 rows × 14 columns

In [222]:

```
df.join(df,on='PassengerID',how='inner', lsuffix='_left', rsuffix='_right')
# SQL-style объединяет столбцы df1 и df2: 'left', 'right', 'outer', 'inner'

#pd.merge(df1, df2,how='left', on='x1')
```

Out[222]:

	PassengerID	PassengerID_left	Name_left	PClass_left	Age_left	Sex_left	Survived_left	SexCode_left	PassengerID_rig
			Allen, Miss						

	PassengerID	PassengerID_left	Name_left	PClass_left	Age_left	Sex_left	Survived_left	SexCode_left	PassengerID_rig
0	1	1	Elisabeth Walton	1st	29.000000	female	1	1	
1	2	2	Allison, Miss Helen Loraine	1st	2.000000	female	0	1	
2	3	3	Allison, Mr Hudson Joshua Creighton	1st	30.000000	male	0	0	
3	4	4	Allison, Mrs Hudson JC (Bessie Waldo Daniels)	1st	25.000000	female	0	1	
4	5	5	Allison, Master Hudson Trevor	1st	0.920000	male	1	0	
...	...	...	...	...	...	...	...	...	
1307	1308	1308	Zabour, Miss Tamini	3rd	30.397989	female	0	1	13
1308	1309	1309	Zakarian, Mr Artun	3rd	27.000000	male	0	0	13
1309	1310	1310	Zakarian, Mr Maprieder	3rd	26.000000	male	0	0	13
1310	1311	1311	Zenni, Mr Philip	3rd	22.000000	male	0	0	13
1311	1312	1312	Lievens, Mr Rene	3rd	24.000000	male	0	0	13

1312 rows × 15 columns

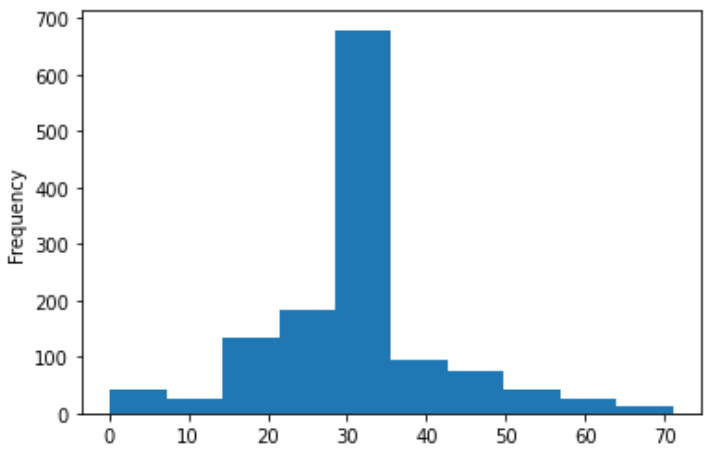
Визуализация

In [226]:

```
df['Age'].plot.hist()
```

Out[226]:

```
<AxesSubplot:ylabel='Frequency'>
```

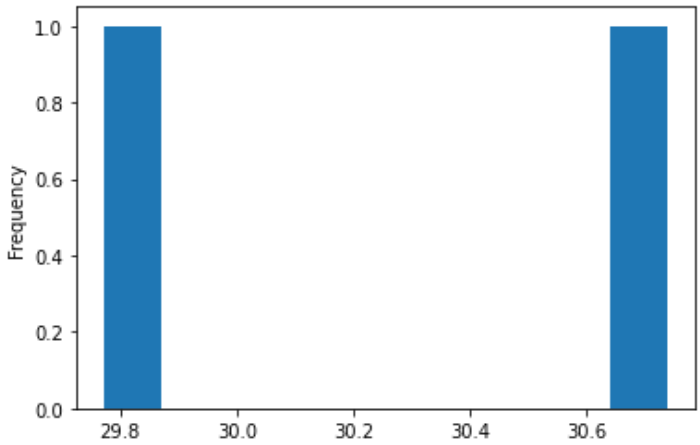


In [258]:

```
d = df.groupby(['Sex'])['Age'].agg(np.mean)
d.plot.hist()
```

Out[258]:

```
<AxesSubplot:ylabel='Frequency'>
```



Операции

In [260]:

```
df + df # add
```

Out[260]:

	PassengerID	Name	PClass	Age	Sex	Survived	SexCode
0	2	Allen, Miss Elisabeth WaltonAllen, Miss Elisab...	1st1st	58.00	femalefemale	2	2
1	4	Allison, Miss Helen LoraineAllison, Miss Helen...	1st1st	4.00	femalefemale	0	2
2	6	Allison, Mr Hudson Joshua CreightonAllison, Mr...	1st1st	60.00	malemale	0	0
3	8	Allison, Mrs Hudson JC (Bessie Waldo Daniels)A...	1st1st	50.00	femalefemale	0	2
4	10	Allison, Master Hudson TrevorAllison, Master H...	1st1st	1.84	malemale	2	0
...	...	...	...	...	...	...	...
1308	2618	Zakarian, Mr ArtunZakarian, Mr Artun	3rd3rd	54.00	malemale	0	0
1309	2620	Zakarian, Mr MapriederZakarian, Mr Maprieder	3rd3rd	52.00	malemale	0	0
1310	2622	Zenni, Mr PhilipZenni, Mr Philip	3rd3rd	44.00	malemale	0	0
1311	2624	Lievens, Mr ReneLievens, Mr Rene	3rd3rd	48.00	malemale	0	0
1312	2626	Zimmerman, LeoZimmerman, Leo	3rd3rd	58.00	malemale	0	0

1313 rows × 7 columns

In [262]:

```
df * df
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~/.local/lib/python3.6/site-packages/pandas/core/ops/array_ops.py in na_arithmetic_op(left, right, op, is_cmp)
    141     try:
--> 142         result = expressions.evaluate(op, left, right)
    143     except TypeError:

~/.local/lib/python3.6/site-packages/pandas/core/computation/expressions.py in evaluate(op, a, b, use_numexpr)
    229         if use_numexpr:
--> 230             return _evaluate(op, op_str, a, b)  # type: ignore
    231     return _evaluate_standard(op, op_str, a, b)
```

```
~/.local/lib/python3.6/site-packages/pandas/core/computation/expressions.py in _evaluate_
standard(op, op_str, a, b)
     67     with np.errstate(all="ignore"):
---> 68         return op(a, b)
     69

TypeError: can't multiply sequence by non-int of type 'str'

During handling of the above exception, another exception occurred:

TypeError                                 Traceback (most recent call last)
<ipython-input-262-71a105fa6e78> in <module>
----> 1 df * df

~/.local/lib/python3.6/site-packages/pandas/core/ops/__init__.py in f(self, other, axis,
level, fill_value)
    649         if isinstance(other, ABCDataFrame):
    650             # Another DataFrame
--> 651             new_data = self._combine_frame(other, na_op, fill_value)
    652
    653         elif isinstance(other, ABCSeries):

~/.local/lib/python3.6/site-packages/pandas/core/frame.py in _combine_frame(self, other,
func, fill_value)
   5861                 return func(left, right)
   5862
-> 5863         new_data = ops.dispatch_to_series(self, other, _arith_op)
   5864         return new_data
   5865

~/.local/lib/python3.6/site-packages/pandas/core/ops/__init__.py in dispatch_to_series(le
ft, right, func, axis)
    273         #  _frame_arith_method_with_reindex
    274
--> 275         bm = left._mgr.operate_blockwise(right._mgr, array_op)
    276         return type(left)(bm)
    277

~/.local/lib/python3.6/site-packages/pandas/core/internals/managers.py in operate_blockwi
se(self, other, array_op)
    362         Apply array_op blockwise with another (aligned) BlockManager.
    363         """
--> 364         return operate_blockwise(self, other, array_op)
    365
    366     def apply(self: T, f, align_keys=None, **kwargs) -> T:

~/.local/lib/python3.6/site-packages/pandas/core/internals/ops.py in operate_blockwise(le
ft, right, array_op)
     36             lvals, rvals = _get_same_shape_values(blk, rblk, left_ea, right_ea)
     37
---> 38             res_values = array_op(lvals, rvals)
     39             if left_ea and not right_ea and hasattr(res_values, "reshape"):
     40                 res_values = res_values.reshape(1, -1)

~/.local/lib/python3.6/site-packages/pandas/core/ops/array_ops.py in arithmetic_op(left,
right, op)
    187     else:
    188         with np.errstate(all="ignore"):
--> 189             res_values = na_arithmetic_op(lvalues, rvalues, op)
    190
    191     return res_values

~/.local/lib/python3.6/site-packages/pandas/core/ops/array_ops.py in na_arithmetic_op(lef
t, right, op, is_cmp)
    147             #  will handle complex numbers incorrectly, see GH#32047
    148             raise
--> 149         result = masked_arith_op(left, right, op)
    150
    151     if is_cmp and (is_scalar(result) or result is NotImplemented):

~/.local/lib/python3.6/site-packages/pandas/core/ops/array_ops.py in masked_arith_op(x, y
```

```
, op)
     89         if mask.any():
     90             with np.errstate(all="ignore"):
---> 91                 result[mask] = op(xrav[mask], yrav[mask])
     92
     93     else:

TypeError: can't multiply sequence by non-int of type 'str'
```