pandas

What is it?

Main library for data analysis in python

Really, must have if you are working with table data

Have 3 main objects - series, dataframe and panel, the latter is not widely used

Series

Series is a numpy array with dict-like indexing essentially
Indices are important for labeling and improved access speed

```
ser = pd.Series([1, 2, 3], index=['A', 'B', 'C'])
A    1
B    2
C    3
dtype: int64
```

Series properties

- homogeneous data type - all elements in a series have the same type, like an array
- sequence - elements in a series are ordered in a sense that their position is constant (untill you chage it), like a list

Series indexing

```
# All the same
ser[0]
1

# By integer location - order, like in list
ser.iloc[0]
1

# By index value - like key in dict
ser.loc['A']
1
```

Other numpy indexing techniques works here too

```
ser[::2]
A    1
C    3
dtype: int64
```

```
ser[[1, 2]]
B    2
C    3
dtype: int64
```

```
ser.loc[ser > 2]
C    3
dtype: int64
```

Underlying structure

All series are just a composition of `np.arrays`. It means that you can obtain data as an arrays and work with them

```
# Data
ser.values
array([1, 2, 3])

# Index object
ser.index
Index(['A', 'B', 'C'], dtype='object')

# Which is essentially an array too
ser.index.values
array(['A', 'B', 'C'], dtype=object)
```

Series methods from arrays

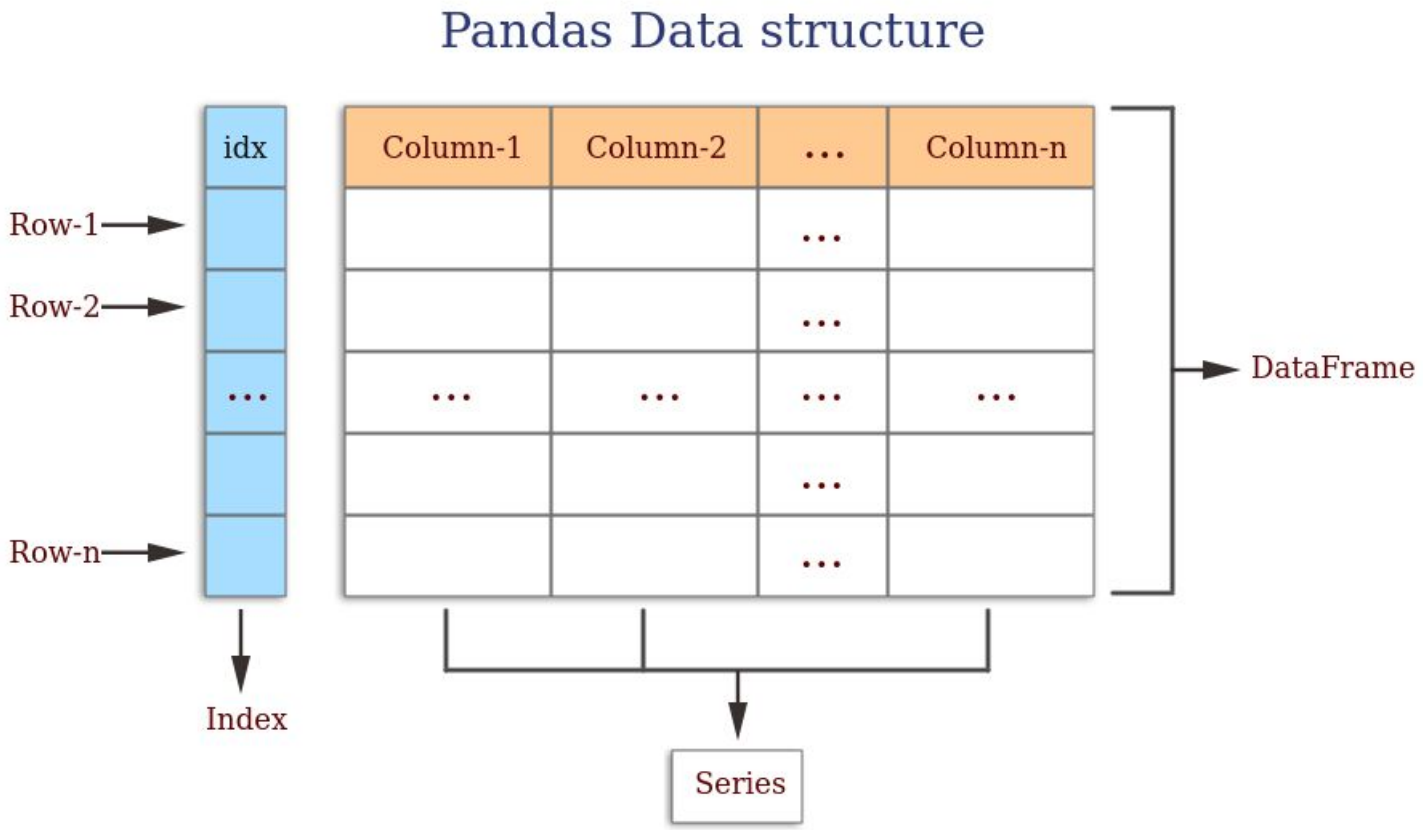
You can use a lion's share of array methods in series

```
ser.max()
3

ser.sum()
6

ser.std()
1.0
```

Special series methods

There are new methods in series, we will talk about them later, in dataframes

DataFrame


Pandas Data structure
idx
Column-1
Column-2
...
Column-n
Row-1
Row-2
Row-n
DataFrame
Index
Series
©v3resource.com

DataFrame morphology

DataFrame is just a 2-dimensional table. It has rows and columns

Usually rows correspond to some observation (e.g. read from sequencing, species, client) while columns correspond to features/attributes/fields of the observation (like read length, species size, client income)

Each column is just a pd.Series with the general index shared by all columns

DataFrame creation

It is possible to create df from the scratch, but usually you will get it from the files with data

```
# One of the easiest ways, though there are much many (from lists, from json)

pd.DataFrame({'a': [1, 2, 3], 'b': [10, 20, 30]})
```

	a	b
0	1	10
1	2	20
2	3	30

Reading files

Most common way how you obtain a dataframe

```
pd.read_csv('/home/arleg/PycharmProjects/bf_course/14.pandas/train.csv')
```

	pos	reads_all	matches	mismatches	deletions	insertions	A	C	T	G	A_fraction	T_fraction	G_fraction
0	279	8045	32	7972	46	8	7911.0	54.0	11.0	NaN	0.983503	0.000815	NaN
1	280	8045	7995	48	19	9	NaN	16.0	13.0	19.0	NaN	0.001355	0.002392
2	281	8045	7983	50	16	5	NaN	24.0	4.0	24.0	NaN	0.000639	0.002875
3	282	8049	7510	505	33	65	105.0	11.0	395.0	NaN	0.012750	0.049096	NaN
4	283	8042	7786	155	115	62	46.0	NaN	93.0	31.0	0.004960	0.010992	0.003186
5	284	8048	7094	104	864	241	24.0	57.0	26.0	NaN	0.002687	0.002429	NaN

read_csv parameters

Returns a dataframe read from the file

- `path` - path to your file, it can be not only csv, but any table with regularly separated values (tsv for instance)
- `sep` - separator between fields, ',' by default
- `header` - index of row which will be used as a column names
- `names` - you can specify appropriate column names
- `index_col` - index of the column in table which will be used as index in a df
- `usecols` - parameter for a list of column names or indices which will be read
- `skiprows` - how many rows should be skipped before reading

Other parameters

- `nrows` - how many rows should be read
- `na_values` - which symbols can be used as NA
- `parse_dates` - whether to parse dates, or list with date columns
- `chunksize` - really handy when data size is too high, after specifying it function will return iterator, generating chunk on each iteration

```
pd.read_csv(path, sep='\t', na_values='?', nrows=10)
```

Other formats

You can read not only csvs, but also these formats

- excel - xls and some other perversions
- clipboard - copied with Ctrl + c content (if it is in tabular format ofc)
- hdf - memory-efficient format
- html - html tables
- pickle - serialized python data
- sql - data from a database

and others

DataFrame exploration

```
# Conventional name for dataframe when you are in a hurry
df =
pd.read_csv('/home/arleg/PycharmProjects/bf_course/14.pandas/
train.csv')
df.head()
```

	pos	reads_all	matches	mismatches	deletions	insertions	A	C	T	G	A_fraction	T_fraction	G_fraction
0	279	8045	32	7972	46	8	7911.0	54.0	11.0	NaN	0.983503	0.000815	NaN
1	280	8045	7995	48	19	9	NaN	16.0	13.0	19.0	NaN	0.001355	0.002392
2	281	8045	7983	50	16	5	NaN	24.0	4.0	24.0	NaN	0.000639	0.002875
3	282	8049	7510	505	33	65	105.0	11.0	395.0	NaN	0.012750	0.049096	NaN
4	283	8042	7786	155	115	62	46.0	NaN	93.0	31.0	0.004960	0.010992	0.003186

```
df.tail(7)
```

	pos	reads_all	matches	mismatches	deletions	insertions	A	C	T	G	A_fraction	T_fraction	G_fraction
56	335	8054	7341	259	459	26	49.0	NaN	63.0	161.0	0.005477	0.006980	0.019948
57	336	8061	7278	367	423	3	97.0	157.0	NaN	128.0	0.011010	NaN	0.015016
58	337	8056	7542	471	58	7	NaN	63.0	47.0	367.0	NaN	0.004584	0.045444
59	338	8061	7967	62	29	5	17.0	NaN	26.0	33.0	0.001060	0.002207	0.003992
60	339	8062	8002	48	14	3	14.0	14.0	33.0	NaN	0.001076	0.002953	NaN
61	340	8061	7918	146	16	4	NaN	75.0	43.0	33.0	NaN	0.005307	0.002951
62	341	8059	7869	192	2	4	NaN	134.0	37.0	17.0	NaN	0.004250	0.002243

Both head and tail accept 1 argument

`n` - number first/last of rows to show

Useful attributes

As you remember these structures are arrays

```
df.values
```

```
array([[2.79000000e+02, 8.04500000e+03, 3.20000000e+01, 7.97200000e+03,
        4.60000000e+01, 8.00000000e+00, 7.91100000e+03, 5.40000000e+01,
        1.10000000e+01,            nan, 9.83502812e-01, 8.15223724e-04,
                   nan, 6.89572250e-03],
```

And we can use many array attributes

```
df.shape
(63, 14)

df.size
882
```

DataFrame index and columns

```
df.index
```

```
RangeIndex(start=0, stop=63, step=1)
```

```
df.columns
```

```
Index(['pos', 'reads_all', 'matches', 'mismatches',
'deletions', 'insertions', 'A', 'C', 'T', 'G',
'A_fraction', 'T_fraction', 'G_fraction',
'C_fraction'], dtype='object')
```

```
# Note this s for many columns
df.dtypes
```

```
pos             int64
reads_all       int64
matches         int64
mismatches      int64
deletions       int64
insertions      int64
A             float64
C             float64
```

This attribute stores a series with column names as index and their type as values

Indexing

Many variants as in R. But we have already covered them

1. `df['column']` - retrieve column from the `df`
2. `df.column` - retrieve column from the `df` if column name is one word and doesn't shade dataframe attributes
3. `df[['a', 'b']]` - retrieve a and b columns from the `df`
4. `df.loc[12]` - retrieve row with the label 12 in the index
5. `df.loc[2:5, ['a', 'b']]` - retrieve rows from the row with label 2 to the row with label 5 inclusively, and a, b columns
6. `df.iloc[12]` - returns a 12th row
7. `df.iloc[:, 3:10:2]` - retrieve all rows with columns from 3rd to 10th with step equal to 2
8. `df[df.pos > 290]` - retrieve all rows where value in pos column greater than 290