Lists [ ]

A list is an ordered an mutable (you can change it) Python container

creating a list: [ ]

```
numbers = [1,2,3,4]
cities = ["Br uge s", " Rom e"]
```

or mix of different types as wel as duplicated elements

list() constructor:

of a string: `list("K ari m")` --> ["K","a","r","i","m"]
of tuple: `list(( " Bru ges ", " Rom e"))` --> ["Bruges", "-Rome"]
of a dictionary `list({ " hyd rog en": 1,"h eli um":2})` --> [hydrogen","helium"]
of a set `list({ " Bru ges ", " Rom e"})` --> ["Bruges", "-Rome"]
of a numpy array `list(n p.a rra y([ 1,2 ,3]))` --> [1,2,3]

accessing:

starting idex = 0; last element = -1
`cities[0]` --> ["Bruges"]
`cities[-2]`--> ["Bruges"]

accessing multiple elements

[start:stop:step]
start index is inclusive
end index is exclusive
default value for step is 1; other values can be omitted = include all

modifying items

replace second item: `cities[1] ="Ge nt"`
replace first two items: `cities[:2] = ["Pa ris " ,"Lo ndo n "]`

Removing elements (del, pop, remove)

del[ ] keyword --> delete first element: `del cities[0]`

list.pop(x) methode: removes the item at the given index, and returns it --> `remove d_c ities = cities.pop(1)`

list.remove(x) methode: deletes the first matching element from a the list, and returns None --> `cities.re mov e("B rug es")`

Inserting elements

list.insert(i,x) --> insert an element x (numbers, booleans, lists) at index i and returns none

list.append(x) --> adds an item to the end of the list - equivalent to list.insert(len(list),x)

Sorting

Lists [ ] (cont)

function: sorted(iterable[, key][, reverse]) --> returns a sorted list => add to variable

methode: list.sort(key=…, reverse=) --> sorts the list in-place

arguments:
- reverse : default = False = ascending
- key: sort a list based on the value returned by the function (def or lambda) provided in the key parameter

Reversing

function: reversed(seq) --> to get a list use the list() constructor ex.: products_reversed = list(reversed(products))

methode: list.reverse() --> reverses the list in-place returning **None**

Concatenate list

+ operator

list.extend(iterable) --> extends the list by appending all the items from the iterable

Check if an element exists in a list

in → Evaluates to True if the object on the left side is included in the object on the right side.
not in → Evaluates to True if the object on the left side is not included in the object on the right side.

basics

import the package

```
import pandas as pd
```

check version

```
pd.__v ers ion__
```

show all rows of dataframe (None = diplay all rows or fill in a number instead)

```
pd.set _op tio n(' dis pla y.m ax_ rows', None)
```

pandas.DataFrame.set_option

Copy data from clipboard

```
df = pd.rea d_c lip board()
```

import data from csv-file (.. = up one level)

```
df = pd.rea d_c sv( " pat h/f ile.cs v")
```

Copy a data frame

basics (cont)

`df_copy = df.copy()`

head() - show first 5 rows (default) or X rows

`df.head()` or `df.hea d(10)`

tail() - show last 5 rows (default) or X rows

`df.tail()` or `df.tai l(10)`

info() - This method prints information about a DataFrame including the index dtype and column dtypes, non-null values and memory usage

`pd.info()`

describe() - Descriptive statistics include those that summarize the central tendency, dispersion and shape of a dataset's distribution, excluding NaN values.

`pd.des cribe()` chain `.round(2)` to clean up the table

pandas.DataFrame.describe

column names

`df.columns`

size of the dataframe

`df.shape`

Quantile() (like describe() but you can define your own values). Default axis = 0 => row-wise

`df.qua nti le( [0.1 ,0.4,0.7, 0.8, 0.9])`

Mean, Standard Deviation, Variance, Count, Median, Min, and Max on column level

`df[" column name"].m ean()` or other function, native or self-made

renaming columns

```
df.ren ame (co lum ns= {'o ldN ame1': 'newNa me1', 'oldNa me2': 'newNa me2'}, inp
lac e=True)
```

Using the argument, **inplace** = True => save dataframe into itself. If we don't state **inplace** = True you need to add result to a new or same dataframe with the "=" operator

reorder columns - pass a list as a list and index

order we want:

```
cols = ['col_ nam e_4', 'col_n ame _2' ,'c ol_ nam e_3', 'col_n ame_1']
```

overwrite the old dataframe with the same dataframe but new column order:

`df= df[cols]`

basics (cont)

adding new columns

```
df[" new _co lum n_n ame "] = ...
```

... = [list] or a function applied to an other column or ...

Count unique rows

`len(df ['c olu mn_ nam e'].un ique()` or `df['column_-name'].nunique

Get count of (unique) values for a particular column

`df.`*`column _name`*`.value _co unts()`

transform dataframe to list

chain with `.tolist()` --> `df.col umn s.t olist()`

making a dataframe

format: df = pd.DataFrame(*data,index values,column names*)

Creating df from list:

```
lst = ['This', 'is', 'a', 'nice', 'cheat', 'sheet']
df1 = pd.Dat aFr ame (lst)
```

Creating df form dict:

```
dict= {"First Column Name":  ["First value", " Secon
}
df2 = pd.Dat aFrame (dict)
```

another example:

```
df3 = pd.Dat aFr ame (np.ra ndo m.r andn(6, 4), inde
```

Index

link: Pandas Index Explained

get index values (strings) - rows ("0", "1", "2" , ...)

`df.index`

>>> RangeIndex(start=0, stop=32561, step=1)

get column index values

`df.columns`

naming index (rows)

`df.ind ex.name = "`*`name_o f_c hoice`*`"`

reset index

`df_new = df.res et_ index()` (the df has already been sliced otherwise the old and new index will be the same)

Index (cont)

Resetting the index will make it a column and recreate another default index

Parameters:

drop = True (default = False) paramater won't create that as column in the dataframe.

inplace = True (default = False)

crosstabs has also index values

```
cross = pd.cro sst ab( df_ new.co l_n ame _1, df_ new.co l_n ame _2)
cross.i ndex
>>>Index( ['v alu e_1 _of _co l_1', 'value _2_ of_ col_1', ...], dtype= 'ob ject', name=' c
ol _na me_1')
```

individual items can be accessed like:

```
cross.l oc ["va lue _?_ of_ col _1"]
```

using the old index (index befor reseting) to access initial dataframe

`df[" col _na me"] [ne w_d f.i nde x_old]` --> index_old (see before name_of_choice where we give our index a name)

Filtering a complementary set from the data

`df_new =df[df.ind ex.i si n(d f_s ub.i ndex)]` --> tilde sign :negate data (True becomes False ...)

df3

	A	B	C	D
a	0.132003	-0.827317	-0.076467	-1.187678
b	1.130127	-1.436737	-1.413681	1.607920
c	1.024180	0.569605	0.875906	-2.211372
d	0.974466	-2.006747	-0.410001	-0.078638
e	0.545952	-1.219217	-1.226825	0.769804
f	-1.281247	-0.727707	-0.121306	-0.097883

Selecting

slicing = getting and setting of subsets of the data set (3 ways)

.loc is primarily label based

.iloc is primarily integer position based

.loc, .iloc, and also [ ] indexing can accept a callable as indexer

`df.loc [ro w_i nde xer ,co lum n_i ndexer]` --> : is the null slice

selecting column(s):

```
df['co lname'] or through a list of columns df[['c oln ame 1', 'co lna me
2']]
```

or directly as an attribute

```
df.colname
```

swapping columns

```
df[['B', 'A']] = df[['A', 'B']]
```

swapping column values on a subset (**you have to swap the raw data !**)

```
df.loc[:, ['B', 'A']] = df[['A', 'B']].to_numpy()
```

Selecting (cont)

create new column A with value 0 --> length of df

```
df['A'] = list(r ang e(l en( df.i nd ex)))
```

slicing using the **[ ]** operator --> [ ] slices the rows

[start:end:step] -->[2:5] --> starts at row 3 (row 2 not included); stops at row 5 (included); default step = 1

If step is negatief = start from the last element

.loc - Selection by label (labels can NOT be integer values)

`df.loc ['i nde x_l abe l_x ':' ind ex_ lab el_y''] --> index_labels are row labels.`

When slicing with .loc **both the start bound AND the stop bound are included**

Select all rows starting from row d, select all columns A to C

`df3.lo c['d':, 'A':'C']` --> red square

getting values with a boolean array

`df3.loc[:, df3.lo c['a'] > 0]` --> all rows and columns where row a >0 --> green square

select numeric columns (column names)

```
df_numeric = df.sel ect _dt ypes(include = [np.nu mbe
])
numeri c_cols = df_num eri c.c olu mns.values
```

select non numeric columns (column names)

```
df_non _nu meric = df.sel ect _dt ypes(exclude=[np.n
mber])
non_nu mer ic_cols = df_non _nu mer ic.c ol umn s.v a
ues
```

setup environment

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplo tli b.p yplot as plt
import matpla tli b.mlab as mlab
import matplotlib
plt.st yle.us e("g gpl ot")
from matplo tli b.p yplot import figure
%matpl otlib inline
matplo tli b.r cPa ram s["f igu re.f ig siz e"] =
(12,8)
pd.opt ion s.m ode.ch ain ed_ ass ignment = None
```

dropping and filling

drop columns

1) focus on columns to keep (add columns to a new dateframe)

```
df_new = df[['c ol_ 1_t o_k eep', 'col_2 _to _keep', ... ]]
```

2) focus on columns to drop

```
df_new = df.dro p([ 'co l_1 _to _dr op' ,'c ol_ 2_t o_d rop ',' col _3_ to_ drop'
, ...], axis=1)
```

fill NaN with some value x

```
df.fil lna(x)
```

datetime

Import statement

```
from datetime import datetime
```
--> python's default library for handling date and time

Creating datetime object

```
dateti me( yea r=2020, month=4, day=11)
```

>>> datetime.datetime(2020, 4, 11, 0, 0)

arguments: year;month;day;hour;minute;second;millisecond

Now()

```
curren t_time = dateti me.n ow()
```

Converting: string to datetime object

```
dateti me.s tr pti me( " 11- 04- 2020, 20:58: 15", " %d- %m-%Y, %H:%M: %S")
```

>>>datetime.datetime(2020, 4, 11, 20, 58, 15)

formating arguments

Converting: datetime to string object

```
dateti me.s tr fti me( dat eti me( yea r=2020, month=4, day=11), " %d/ %m/ %
Y")
```

>>> '11/04/2020'

Data range in Pandas

```
pd.dat e_r ang e(s tar t=d ate tim e(y ear =2020, month=4, day=11 ),p eri od
s =3, fre q='D')
```

>>>DatetimeIndex(['2020-04-11', '2020-04-12', '2020-04-13'], dtype='datetime64[ns]', freq='D')

frequency aliases

start argument can also be like: '2020-04-11' or '2020/04/11' or '2020, may 11'

apply function

under construction

missing data

heatmap

```
cols = df.col umn s[:30]
```
--> select first 30 columns (names)

```
colours = ["bl ue", " yel low "]
```
--> missing data will be disp

```
sns.he atm ap( df[ col s].i sn ull (), cma p=s ns.c
```

false's. If value is NA then isnull() returns true = 1

data percentage list

```
missing = {}
for col in df.col umns:
 pct_mi ssing = np.mea n(d f[c ol].is null())
 missin g[col] = round( pct _mi ssi ng*100)
missin g_s orted = {key: value for key, value in sort
se =True)}
```