Set and mapping types (unordered):

```
set         {'Fred', 'John'}  set(('Fred', 'John'))
frozenset   frozenset(('Fred', 'John'))
dict        {'Fred': 123, 42: 'John'}
            dict([('Fred', 123), (42, 'John')])
            dict(Fred=123, John=42)
```

Immutable set methods and operations:

```
intersection()  symmetric_difference()  issubset()
union()         copy()                  issuperset()
difference()                            isdisjoint()

{1, 2} & {2, 3} == {2}            {1, 2} == {2, 1}
{1, 2} | {2, 3} == {1, 2, 3}          {1} < {1, 2}
{1, 2} - {2, 3} == {1}            {1, 2} <= {1, 2}
{1, 2} ^ {2, 3} == {1, 3}
```

Set mutation methods:

```
add()    update()    intersection_update()
pop()    remove()    difference_update()
clear()  discard()   symmetric_difference_update()
```

Mapping methods and operations:

```
get()         keys()    pop()       copy()
setdefault()  values()  popitem()   fromkeys()
update()      items()   clear()

x = {'a': 1, 'b': 2};   x['d'] = 5
'b' in x == True;       x['a'] == 1;   del x['b']
```

List and dict comprehensions:

```
[2 * i for i in range(3)] == [0, 2, 4]
{i: i ** 2 for i in range(3)}
         == {0: 0, 1: 1, 2: 4}
```

Functions

Simple function definition, takes an argument of any type:

```
def double(x):          |   double(2) == 4
    return x * 2        |   double('abc') == 'abcabc'
```

Function that does not explicitly return a value:

```
def idle(): pass        |   idle() == None
```

Function with optional arguments:

```
def multiply(x, y=2):          multiply(3) == 6
    return x * y             multiply(3, 5) == 15
                            multiply(3, y=5) == 15
```

Classes

Simple class definition with attributes and constructor:

```
class Simple:                  |  obj = Simple(7)
    x = None                   |  obj.x == 7
    def __init__(self, x):
        self.x = x
```

Subclass which accesses a method of its Superclass:

```
class XY(Simple):              |  obj = XY(7, 9)
    y = None                   |  obj.x == 7
    def __init__(self, x, y):  |  obj.y == 9
        super().__init__(x)
        self.y = y
```

Class with a method that can be called on instances:

```
class CalcZ(XY):               |  obj = CalcZ(7, 9)
    def do_z(self):            |  obj.do_z() == 63
        return self.x * self.y
```

Class with an automatically computed attribute:

```
class AutoZ(XY):               |  obj = AutoZ(7, 9)
    @property                  |  obj.z == 63
    def z(self):
        return self.x * self.y
```

This cheat sheet refers to Python 3.7:
https://docs.python.org/3.7/

Coding style conventions according to PEP8
https://python.org/dev/peps/pep-0008/

https://www.cusy.io/de/seminare

>CUSY_

Python cheat sheet

Code structure

Grouping: Whitespace has meaning. Line breaks separate statements, indentation creates logical blocks. Comments run from # to line break. Functional units go into modules (files) and packages (directories); each source file imports any modules it uses:

```
import math
for x in range(0, (4+1)*2):  # numbers 0 <= x < 10
    y = math.sqrt(x)
    print('The square root of {} is {}'.format(
            x, y))
```

Variable names: May contain letters (unicode, case-sensitive), numerals and _.

Logic and flow control

Conditions: compound statement or expression:

```
if x < y:                      |  print('equal'
    print(x)                   |      if x == y
elif x > y:                    |      else 'unequal')
    print(y)
else:
    print('equal')
```

Iteration: over sets or until termination:

```
for name in ['John', 'Fred', 'Bob']:
    if name.startswith('F'):
        continue
    print(name)

while input('Stop?') != 'stop':
    if 'x' in input('Do not type an x.'):
        print('You typed an x.')
        break
else:
    print('Loop finished without typing an x.')
```

Data export

Data as NumPy array:
```
df.values
```

Save data as CSV file:
```
df.to_csv('output.csv', sep=",")
```

Format a data frame as tabular string:
```
df.to_string()
```

Convert a data frame to a dictionary:
```
df.to_dict()
```

Save a data frame as an Excel table:
```
df.to_excel('output.xlsx')
```

(requires package xlwt)

Visualization

Import matplotlib:
```
import pylab as plt
```

Start a new diagram:
```
plt.figure()
```

Scatter plot:
```
df.plot.scatter('col1', 'col2', style='ro')
```

Bar plot:
```
df.plot.bar(x='col1', y='col2', width=0.7)
```

Area plot:
```
df.plot.area(stacked=True, alpha=1.0)
```

Box-and-whisker plot:
```
df.plot.box()
```

Histogram over one column:
```
df['col1'].plot.hist(bins=3)
```

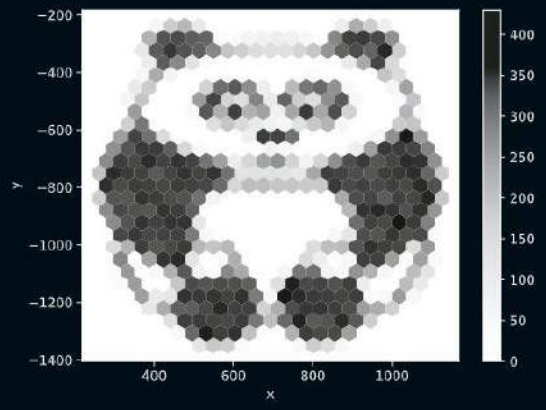
Histogram over all columns:
```
df.plot.hist(bins=3, alpha=0.5)
```

Set tick marks:
```
labels = ['A', 'B', 'C', 'D']
positions = [1.0, 2.0, 3.0, 4.0]
plt.xticks(positions, labels)
plt.yticks(positions, labels)
```

Select area to plot:
```
plt.axis([0.0, 2.5, 0.0, 10.0])
# [from x, to x, from y, to y]
```

Label diagram and axes:
```
plt.title('Correlation')
plt.xlabel('Nunstück')
plt.ylabel('Slotermeyer')
```

Save most recent diagram:
```
plt.savefig('plot.png')
plt.savefig('plot.png', dpi=300)
plt.savefig('plot.svg')
```



https://www.cusy.io/de/seminare

>CUSY_

Pandas cheat sheet

All of the following code examples refer to this table:

df=

	col1	col2
A	1	4
B	2	5
C	3	6

Getting started

Import pandas:
```
import pandas as pd
```

Create a series:
```
s = pd.Series([1, 2, 3], index=['A', 'B', 'C'],
              name='col1')
```

Create a data frame:
```
data = [[1, 4], [2, 5], [3, 6]]
index = ['A', 'B', 'C']
df = pd.DataFrame(data, index=index,
                  columns=['col1', 'col2'])
```

Load a data frame:
```
df = pd.read_csv('filename.csv',
                 sep=',',
                 names=['col1', 'col2'],
                 index_col=0,
                 encoding='utf-8',
                 nrows=3)
```

Selecting rows and columns

Select single column:
```
df['col1']
```

Git cheat sheet

Git installation

For GNU/Linux distributions, Git should be available in the standard system repository. For example, in Debian/ Ubuntu please type

```
$ sudo apt-get install git
```

Git config

```
$ git config --global user.name "My Name"
$ git config --global user.email "me@cusy.io"
```

Set name and email address that will be attached to your commits and tags.

```
$ git config --global color.ui auto
```

Set colorisation of Git output

.gitignore

Some files usually shouldn't be tracked by git. They are written to a special file named `.gitignore`. You can find helpful templates at `github.com/veit/dotfiles/`.

Start a project

```
$ git init [my_project]
```

Create a new local repository.

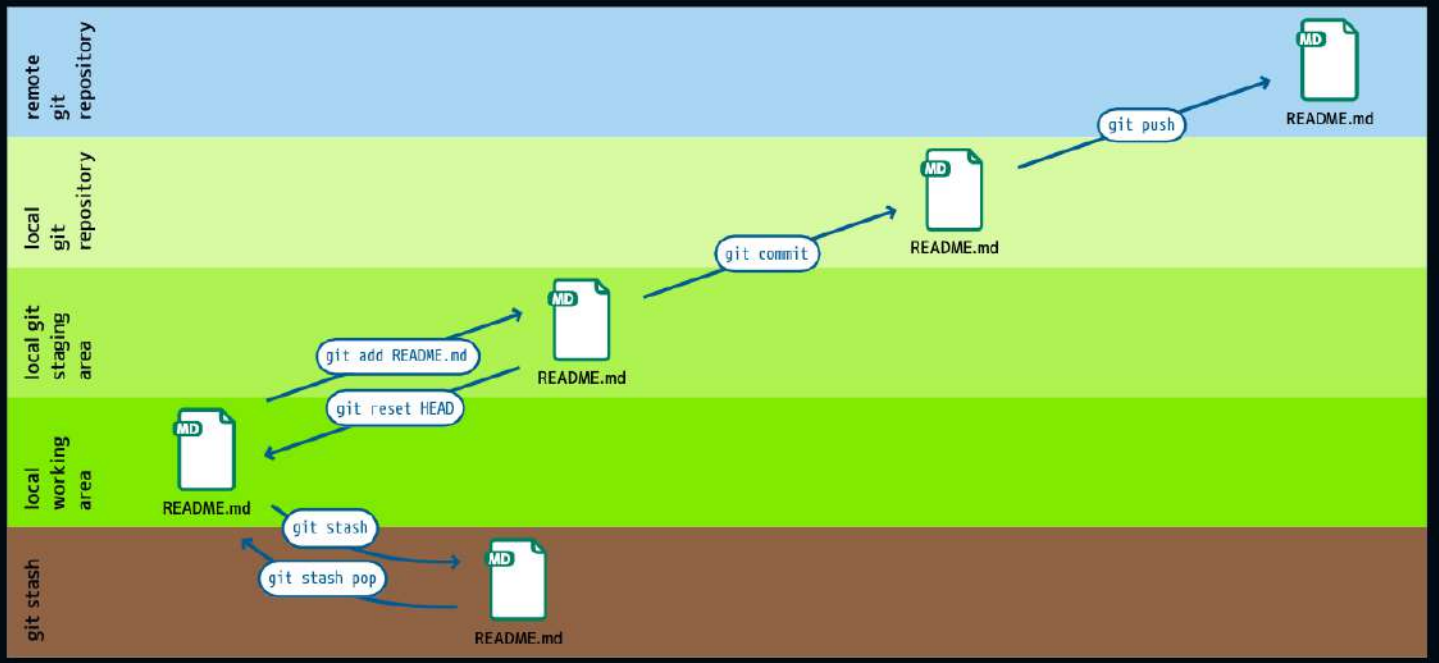
If `[my_project]` is provided, Git will create a new directory and initialise it as a repository.

If `[my_project]` is not provided, the new repository is initialised in the current directory.

```
$ git clone [project_url]
```

Downloads a project with all branches and the entire history from the remote repository.

Work on a project




```
$ git status
```

Display the status of the working directory with new, staged and modified files for the current branch.

```
$ git add [file]
```

Add a file to the staging area.

```
$ git add -p [file]
```

Add only parts of a file to the staging area.

```
$ git diff [file]
```

Show changes between working and staging area.

```
$ git diff --staged [file]
```

Show changes between staging area and repository.

```
$ git checkout -- [file]
```

Irrevocably discard changes in the working directory.

```
$ git commit -m 'Commit message'
```

Create a new commit from added changes.

```
$ git reset [file]
```

Revert the file to the last commited version.

```
$ git rm [file]
```

Remove the file from working directory and staging area.

```
$ git stash
```

Put current changes from your working directory into stash for later use.

```
$ git stash list
```

List the modifications stashed away with `git stash`.

```
$ git stash show [<stash>]
```

List the modifications stashed away with `git stash`.