

Importing Text Files I

```
open(f ile _name, 'r') # open the file
file.r ead() # read the file
file.c lose() # close the file
file.c losed() # check if the file is closed
```

It is a good practice to close the file after reading it when using 'open'

Importing Text Files II

```
with open(f ile _name) as file:
    # open the file

file.r ead() # read the file

file.readline() # read line by line
```

When using the 'with' statement there is no need to close the file

Importing Flat Files with Numpy I

```
import numpy as np

np.loa dtx t(f ile _name, delimiter=' ')

skiprows=1

usecol s=[0, 2]
```

`dtype = str`

loadtxt only works with numeric data

Importing Flat Files with Numpy II

```
import numpy as np

np.rec fro mcs v(file, delimi ter =",", names= True, dtype = None)

np.gen fro mt x t(file, delimi ter =",", names= True, dtype = None)
```

with the functions recfromcsv() and genfromtxt() we are able to import data with different types

Importing Stata Files

```
import pandas as pd

df = pd.rea d_s tat a(' dis are a .dta')
```

Importing Flat Files With Pandas

```
import pandas as pd # import pandas

pd.rea d_c sv( file # open csv file

nrows=5 # argument for the number of rows to load
```

```
import # argument for no header
numpy # argument to set delimiter

import # argument
the file # argument
# argument # '
to skip a
specific
row # argument
# argument
to only
show # argument
# argument
specific
columns # argument
# argument
to import
the data
as string
```

Import pickled files

```
import pickle # import
numpy # numpy

with open(f ile _name, 'r') as fi # open
the file # open
the file # the file

pickle.lo ad( file) # the file
```

Importing Spreadsheet Files

```
import pandas as pd # importing
pandas # pandas

pd.Exc elF ile (file) # opening
the file # opening
the file # the file

xl.she et_ names # exporting
the shee # the shee
names # names

xl.par se( she et_ nam e/i # loading
ndex) # sheet to
datafram # datafram

skipro ws= [index] # skipping
specific # specific
row # row

names= [List of Names] # naming
the # the
sheet's # sheet's
columns # columns

usecol s=[0, ] # parse
specific # specific
columns # columns
```

skiprows, names and usecols are all arguments of the function parse()

Importing SAS Files

```
from sas7bdat import SAS7BDAT

import pandas as pd

with SAS7BD AT( 'fi le_ name') as
file:

file.t o_d ata _fr ame()
```

Importing HDF5 files

```
import numpy as np      import numpy
import h5py              importing the
                          h5py library
h5py.File(file, 'r')    reading the
                          file
```

Importing MATLAB files

```
import scipy.io          importing
                          scipy.io
scipy.io.loadmat('file_ reading
name')                   the file
```

Relational databases I

```
import pandas as pd
from sqlalchemy import create_engine
```

```
engine = create_engine('database://name:password@host:port/database')
con = engine.connect()
```

```
rs = con.execute('SELECT * FROM Album')
```

```
df = pd.DataFrame(rs.fetchall())
```

```
df.columns = rs.keys
```

```
con.close()
```

The best practice is to close the connection

Relational databases II

```
engine = create_engine('database://name:password@host:port/database')
with engine.connect() as con:
```

```
rs = con.execute('sql code')
```

```
df = pd.DataFrame(rs.fetchmany(size=3))
```

With 'open' you don't have to close the connection at the end

Relational databases III

```
engine = create_engine('database://name:password@host:port/database')
df = pd.read_sql_query('SQL code', engine)
```

Fastest way to connect to a database engine and perform query

importing pandas
importing the necessary library

creating an engine
connecting to the engine
perform query

save as a dataframe

set columns names

close the connection



By **issamdbd**
cheatography.com/issamdbd/

Published 16th August, 2019.
Last updated 16th August, 2019.
Page 2 of 2.

Sponsored by **Readable.com**
Measure your website readability!
<https://readable.com>