Cheatography

Importing Data in Python I Cheat Sheet

by issambd via cheatography.com/88527/cs/20287/

Importing Text Files I

`open(file_name, 'r')`	open the file
`file.read()`	read the file
`file.close()`	close the file
`file.closed()`	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(file_name) as file :`	open the file
`file.read()`	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`	import numpy
`np.loadtxt(file_name, delimiter= ' ')`	importing the file
`skiprows=1`	argument to skip a specific row
`usecols=[0, 2]`	argument to only show specific columns
`dtype = str'	argument to import the data as string

loadtxt only works with numeric data

Importing Flat Files with Numpy II

`import numpy as np`	import numpy
`np.recfromcsv(file, delimiter=",", names=-True, dtype=None)`	open the file
`np.genfromtxt(file, delimiter=',', names=-True, dtype=None)`	open the file

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

Importing Stata Files

`import pandas as pd`	importing pandas
`df = pd.read_stata('-disarea.dta')`	reading the stata file

Importing Flat Files With Pandas

`import pandas as pd`	import pandas
`pd.read_csv-(file)`	open csv file
`nrows=5`	argument for the number of rows to load
`header=None`	argument for no header
`sep='\t'`	argument to set delimiter
`comment='#'`	argument takes characters that comments occur after in the file
`na_values='-Nothing'`	argument to recognize a string as a NaN Value

Import pickled files

`import pickle`	import the library
`with open(file_name, 'rb') as file :`	open file
`pickle.load(file)`	read file

Importing Spreadsheet Files

`import pandas as pd`	importing pandas
`pd.ExcelFile(file)`	opening the file
`xl.sheet_names`	exporting the sheet names
`xl.parse(sheet-_name/index)`	loading a sheet to a dataframe
`skiprows=[index]`	skipping a specific row
`names=[List of Names]`	naming the sheet's columns
`usecols=[0,]`	parse spesific columns

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

Importing SAS Files

`from sas7bdat import SAS7BDAT`	importing sas7bdat library
`import pandas as pd`	importing pandas
`with SAS7BDAT('file_name') as file:`	opening the file
`file.to_data_frame()`	loading the file as dataframe

By **issambd**
cheatography.com/issambd/

Published 16th August, 2019.
Last updated 16th August, 2019.

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
`cipy.io.loadmat('-file_name')`	reading the file

Relational databases I

`import pandas as pd`	importing pandas
`from sqlalchemy import create_engine`	importing the necessary library
`engine = create_engine('databasetype:///name.databasetype')`	creating an engine
`con = engine.connect()`	connecting to the engine
`rs = con.execute('SELECT * FROM Album')`	performe query
`df = pd.DataFrame(rs.fetchall())`	save as a dataframe
`df.columns = rs.keys`	set columns names
`con.close()`	close the connection

The best practice is to close the connection

Relational databases II

`engine = create_engine('databasetype:///name.databasetype')`	creating an engine
`with engine.connect() as con:`	connecting to the engine
`rs = con.execute('sql code')`	performe query
`df = pd.DataFrame(rs.fetchmany(size=3))`	load a number of rows as a dataframe

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

Relational databases III

`engine = create_engine('databasetype:///name.databasetype')`	creating an engine
`df = pd.read_sql_query('SQL code', engine)`	performe query

Fastest way to connect to a database and perform query