

Module	Description	Example	Script
collections	defaultdict, creating for lists	<code>by_zone = defaultdict(list)</code>	<code>g10/demo.py</code>
collections	defaultdict, importing	<code>from collections import defaultdict</code>	<code>g10/demo.py</code>
core	dictionary, adding a new entry	<code>co['po'] = 'CO'</code>	<code>g05/demo.py</code>
core	dictionary, checking for existing key	<code>if fips in name_by_fips:</code>	<code>g09/demo.py</code>
core	dictionary, creating	<code>co = {'name':'Colorado', 'capital':'Denver'}</code>	<code>g05/demo.py</code>
core	dictionary, deleting an entry	<code>del name_by_fips["00"]</code>	<code>g09/demo.py</code>
core	dictionary, iterating over keys	<code>for fips in name_by_fips.keys():</code>	<code>g09/demo.py</code>
core	dictionary, iterating over values	<code>for rec in name_by_fips.values():</code>	<code>g09/demo.py</code>
core	dictionary, looking up a value	<code>name = ny['name']</code>	<code>g05/demo.py</code>
core	dictionary, making a list of	<code>list1 = [co,ny]</code>	<code>g05/demo.py</code>
core	dictionary, obtaining a list of keys	<code>names = super_dict.keys()</code>	<code>g05/demo.py</code>
core	dictionary, sorting keys	<code>for tz in sorted(by_zone.keys()):</code>	<code>g10/demo.py</code>
core	f-string, using a formatting string	<code>print(f"PV of {payment} with T={year} and r={r} is \${p. . .</code>	<code>g07/demo.py</code>
core	file, closing	<code>fh.close()</code>	<code>g02/demo.py</code>
core	file, opening for reading	<code>fh = open('states.csv')</code>	<code>g05/demo.py</code>
core	file, opening for writing	<code>fh = open(filename,"w")</code>	<code>g02/demo.py</code>
core	file, output using print	<code>print("It was written during",year,file=fh)</code>	<code>g02/demo.py</code>
core	file, output using write	<code>fh.write("Where was this file was written?\n")</code>	<code>g02/demo.py</code>
core	file, print without adding spaces	<code>print('\nOuter:\n', join_o['_merge'].value_counts(), s. . .</code>	<code>g15/demo.py</code>
core	file, reading one line at a time	<code>for line in fh:</code>	<code>g05/demo.py</code>
core	for, looping through a list	<code>for n in a_list:</code>	<code>g04/demo.py</code>
core	function, calling	<code>d1_ssqr = sumsq(d1)</code>	<code>g06/demo.py</code>
core	function, calling with an optional argument	<code>sample_function(100, 10, r=0.07)</code>	<code>g07/demo.py</code>
core	function, defining	<code>def sumsq(values):</code>	<code>g06/demo.py</code>
core	function, defining with optional argument	<code>def sample_function(payment,year,r=0.05):</code>	<code>g07/demo.py</code>
core	function, returning a result	<code>return values</code>	<code>g06/demo.py</code>
core	if statement, testing for equality	<code>if fips == "36":</code>	<code>g09/demo.py</code>
core	list, appending an element	<code>a_list.append("four")</code>	<code>g03/demo.py</code>
core	list, create via comprehension	<code>cubes = [n**3 for n in a_list]</code>	<code>g04/demo.py</code>
core	list, creating	<code>a_list = ["zero","one","two","three"]</code>	<code>g03/demo.py</code>

Module	Description	Example	Script
core	list, determining length	<code>n = len(b_list)</code>	g03/demo.py
core	list, extending with another list	<code>a_list.extend(a_more)</code>	g03/demo.py
core	list, generating a sequence	<code>b_list = range(1,6)</code>	g04/demo.py
core	list, joining with spaces	<code>a_string = " ".join(a_list)</code>	g03/demo.py
core	list, selecting an element	<code>print(a_list[0])</code>	g03/demo.py
core	list, selecting elements 0 to 3	<code>print(a_list[:4])</code>	g03/demo.py
core	list, selecting elements 1 to 2	<code>print(a_list[1:3])</code>	g03/demo.py
core	list, selecting elements 1 to the end	<code>print(a_list[1:])</code>	g03/demo.py
core	list, selecting last 3 elements	<code>print(a_list[-3:])</code>	g03/demo.py
core	list, selecting the last element	<code>print(a_list[-1])</code>	g03/demo.py
core	list, sorting	<code>c_sort = sorted(b_list)</code>	g03/demo.py
core	list, summing	<code>tot_inc = sum(incomes)</code>	g08/demo.py
core	math, raising a number to a power	<code>a_cubes.append(n**3)</code>	g04/demo.py
core	math, rounding a number	<code>rounded = round(ratio,2)</code>	g05/demo.py
core	string, concatenating	<code>name = s1+" "+s2+" "+s3</code>	g02/demo.py
core	string, converting to an int	<code>values.append(int(line))</code>	g06/demo.py
core	string, converting to title case	<code>name = codes[key].title()</code>	g11/demo.py
core	string, creating	<code>filename = "demo.txt"</code>	g02/demo.py
core	string, including a newline character	<code>fh.write(name+"!\n")</code>	g02/demo.py
core	string, splitting on a comma	<code>parts = line.split(',')</code>	g05/demo.py
core	string, splitting on whitespace	<code>b_list = b_string.split()</code>	g03/demo.py
core	string, stripping blank space	<code>clean = [item.strip() for item in parts]</code>	g05/demo.py
core	tuple, creating	<code>this_tuple = (med_density,state)</code>	g10/demo.py
core	tuple, creating via split	<code>(last,first) = name.split(',')</code>	g11/demo.py
core	tuple, looping over	<code>for (den,state) in sorted(by_density):</code>	g10/demo.py
core	tuple, sorting	<code>for key in sorted(codes):</code>	g11/demo.py
core	tuple, testing equality of	<code>if key == (29,'VA'):</code>	g11/demo.py
csv	opening a file for use with DictWriter	<code>fh = open(outfile,'w',newline="")</code>	g09/demo.py
csv	setting up a DictReader object	<code>reader = csv.DictReader(fh)</code>	g08/demo.py
csv	setting up a DictWriter object	<code>writer = csv.DictWriter(fh,fields)</code>	g09/demo.py
csv	using DictReader with a list	<code>reader = csv.DictReader(lines)</code>	g10/demo.py
csv	writing a header with DictWriter	<code>writer.writeheader()</code>	g09/demo.py
csv	writing a record with DictWriter	<code>writer.writerow(name_rec)</code>	g09/demo.py

Module	Description	Example	Script
io	converting a byte stream to characters	<code>inp_handle = io.TextIOWrapper(inp_byte)</code>	g11/demo.py
json	importing the module	<code>import json</code>	g05/demo.py
json	using to print an object nicely	<code>print(json.dumps(list1,indent=4))</code>	g05/demo.py
matplotlib	axes, setting a title	<code>ax1.set_title('Population')</code>	g13/demo.py
matplotlib	axis, labeling X axis	<code>ax1.set_xlabel('Millions')</code>	g13/demo.py
matplotlib	figure, saving	<code>fig1.savefig('figure.png')</code>	g13/demo.py
matplotlib	figure, tuning the layout	<code>fig1.tight_layout()</code>	g13/demo.py
matplotlib	importing pyplot	<code>import matplotlib.pyplot as plt</code>	g13/demo.py
matplotlib	setting a figure title	<code>fig1.suptitle('Electric Power Plants in Onondaga and Os. . .</code>	g16/demo.py
matplotlib	setting the default resolution	<code>plt.rcParams['figure.dpi'] = 300</code>	g18/demo.py
matplotlib	using subplots to set up a figure	<code>fig1, ax1 = plt.subplots()</code>	g13/demo.py
numpy	computing a median	<code>med_density = round(np.median(this_list), 2)</code>	g10/demo.py
numpy	importing	<code>import numpy as np</code>	g10/demo.py
pandas	columns, dividing with explicit alignment	<code>normed2 = 100*states.div(pa_row,axis='columns')</code>	g12/demo.py
pandas	columns, listing names	<code>print('\nColumns:', list(states.columns))</code>	g12/demo.py
pandas	columns, renaming	<code>county = county.rename(columns={'B01001_001E':'pop'})</code>	g14/demo.py
pandas	columns, retrieving one by name	<code>pop = states['pop']</code>	g12/demo.py
pandas	columns, retrieving several by name	<code>print(pop[some_states]/1e6)</code>	g12/demo.py
pandas	dataframe, appending	<code>gen_all = pd.concat([gen_oswego, gen_onondaga])</code>	g16/demo.py
pandas	dataframe, dropping a column	<code>both = both.drop(columns='_merge')</code>	g16/demo.py
pandas	dataframe, dropping duplicates	<code>flood = flood.drop_duplicates(subset='TAX_ID')</code>	g15/demo.py
pandas	dataframe, finding duplicate records	<code>dups = parcels.duplicated(subset='TAX_ID', keep=False . . .</code>	g15/demo.py
pandas	dataframe, inner 1:1 merge	<code>join_i = parcels.merge(flood,</code>	g15/demo.py
pandas	dataframe, left 1:1 merge	<code>join_l = parcels.merge(flood,</code>	g15/demo.py
pandas	dataframe, left m:1 merge	<code>both = gen_all.merge(plants,</code>	g16/demo.py
pandas	dataframe, making a copy	<code>subset_copy = sample[keepvars].copy()</code>	g17/demo.py
pandas	dataframe, outer 1:1 merge	<code>join_o = parcels.merge(flood,</code>	g15/demo.py
pandas	dataframe, reading zipped pickle format	<code>sample2 = pd.read_pickle('sample_pkl.zip')</code>	g17/demo.py
pandas	dataframe, resetting the index	<code>hourly = hourly.reset_index()</code>	g18/demo.py
pandas	dataframe, right 1:1 merge	<code>join_r = parcels.merge(flood,</code>	g15/demo.py
pandas	dataframe, saving in zipped pickle format	<code>sample.to_pickle('sample_pkl.zip')</code>	g17/demo.py

Module	Description	Example	Script
pandas	dataframe, selecting rows via boolean	<code>dup_rec = flood[dups]</code>	<code>g15/demo.py</code>
pandas	dataframe, selecting rows via query	<code>ngcc = both.query("Technology == 'Natural Gas Fired Com. . .</code>	<code>g16/demo.py</code>
pandas	dataframe, sorting by a column	<code>county = county.sort_values('pop')</code>	<code>g14/demo.py</code>
pandas	dataframe, sorting by index	<code>summary = summary.sort_index(ascending=False)</code>	<code>g16/demo.py</code>
pandas	dataframe, unstacking an index level	<code>bymo = bymo.unstack('month')</code>	<code>g18/demo.py</code>
pandas	datetime, building via <code>to_datetime()</code>	<code>date = pd.to_datetime(recs['ts'])</code>	<code>g15/demo.py</code>
pandas	datetime, building with a format	<code>ymd = pd.to_datetime(sample['TRANSACTION_DT'], format=. . .</code>	<code>g17/demo.py</code>
pandas	datetime, extracting day attribute	<code>recs['day'] = date.dt.day</code>	<code>g15/demo.py</code>
pandas	datetime, extracting hour attribute	<code>recs['hour'] = date.dt.hour</code>	<code>g15/demo.py</code>
pandas	displaying all columns	<code>pd.set_option('display.max_columns',None)</code>	<code>g17/demo.py</code>
pandas	displaying all rows	<code>pd.set_option('display.max_rows', None)</code>	<code>g12/demo.py</code>
pandas	groupby, counting records via size	<code>summary['units'] = tech_by_kv.size()</code>	<code>g16/demo.py</code>
pandas	groupby, summing a variable	<code>state = county.groupby('state')['pop'].sum()</code>	<code>g14/demo.py</code>
pandas	groupby, using with one grouping variable	<code>by_reg = state_data.groupby('Region')</code>	<code>g13/demo.py</code>
pandas	importing the module	<code>import pandas as pd</code>	<code>g12/demo.py</code>
pandas	index, creating with two-levels	<code>county = county.set_index(['state','county'])</code>	<code>g14/demo.py</code>
pandas	index, listing names	<code>print('\nIndex (rows):', list(states.index))</code>	<code>g12/demo.py</code>
pandas	index, renaming values	<code>div_pop = div_pop.rename(index=div_names)</code>	<code>g13/demo.py</code>
pandas	index, retrieving a row by name	<code>pa_row = states.loc['Pennsylvania']</code>	<code>g12/demo.py</code>
pandas	index, retrieving first rows by location	<code>print(low_to_high.iloc[0:10])</code>	<code>g12/demo.py</code>
pandas	index, retrieving last rows by location	<code>print(low_to_high.iloc[-5:])</code>	<code>g12/demo.py</code>
pandas	index, setting to a column	<code>new_states = states.set_index('name')</code>	<code>g12/demo.py</code>
pandas	index, setting to a column in place	<code>states.set_index('name',inplace=True)</code>	<code>g12/demo.py</code>
pandas	plotting, bar plot	<code>reg_pop.plot.bar(ax=ax1)</code>	<code>g13/demo.py</code>
pandas	plotting, disabling legend	<code>summary.plot.barh(y='mw',ax=ax1,legend=None)</code>	<code>g16/demo.py</code>
pandas	plotting, horizontal bar plot	<code>div_pop.plot.barh(ax=ax1)</code>	<code>g13/demo.py</code>
pandas	reading, csv data	<code>states = pd.read_csv('state-data.csv')</code>	<code>g12/demo.py</code>
pandas	reading, csv using dtype	<code>geocodes = pd.read_csv('state-geocodes.csv',dtype=str)</code>	<code>g13/demo.py</code>
pandas	series, converting strings to title case	<code>fixname = subset_view['NAME'].str.title()</code>	<code>g17/demo.py</code>

Module	Description	Example	Script
pandas	series, converting to float	<code>sample['dollars'] = sample['TRANSACTION_AMT'].astype(fl. . .</code>	<code>g17/demo.py</code>
pandas	series, element-by-element or	<code>is_either = is_ca is_tx</code>	<code>g17/demo.py</code>
pandas	series, retrieving an element	<code>print("\nFlorida's population:", pop['Florida']/1e6)</code>	<code>g12/demo.py</code>
pandas	series, sorting by value	<code>low_to_high = normed['med_pers_inc'].sort_values()</code>	<code>g12/demo.py</code>
pandas	series, summing	<code>reg_pop = by_reg['pop'].sum()/1e6</code>	<code>g13/demo.py</code>
pandas	series, unstacking	<code>tot_wide = tot_amt.unstack('PGI')</code>	<code>g17/demo.py</code>
pandas	series, using <code>isin()</code>	<code>fixed = flood['TAX_ID'].isin(dup_rec['TAX_ID'])</code>	<code>g15/demo.py</code>
pandas	series, using <code>value_counts()</code>	<code>print('\nOuter:\n', join_o['_merge'].value_counts(), s. . .</code>	<code>g15/demo.py</code>
pandas	using <code>qcut</code> to create deciles	<code>dec = pd.qcut(county['pop'], 10, labels=range(1,11))</code>	<code>g14/demo.py</code>
pandas	using <code>xs</code> to select from an index	<code>print(county.xs('04',level='state'))</code>	<code>g14/demo.py</code>
requests	calling the <code>get()</code> method	<code>response = requests.get(api,payload)</code>	<code>g19/demo.py</code>
requests	checking the URL	<code>print('url:', response.url)</code>	<code>g19/demo.py</code>
requests	checking the response text	<code>print(response.text)</code>	<code>g19/demo.py</code>
requests	checking the status code	<code>print('status:', response.status_code)</code>	<code>g19/demo.py</code>
requests	decoding a JSON response	<code>rows = response.json()</code>	<code>g19/demo.py</code>
requests	importing the module	<code>import requests</code>	<code>g19/demo.py</code>
scipy	calling newton's method	<code>cr = opt.newton(find_cube_root,xinit,maxiter=20,args=[y. . .</code>	<code>g07/demo.py</code>
scipy	importing the module	<code>import scipy.optimize as opt</code>	<code>g07/demo.py</code>
seaborn	adding a title to a grid object	<code>jg.fig.suptitle('Distribution of Hourly Load')</code>	<code>g18/demo.py</code>
seaborn	barplot	<code>sns.barplot(data=hourly,x='hour',y='usage',hue='month',. . .</code>	<code>g18/demo.py</code>
seaborn	basic violin plot	<code>sns.violinplot(data=janjul,x="month",y="usage")</code>	<code>g18/demo.py</code>
seaborn	boxenplot	<code>sns.boxenplot(data=janjul,x="month",y="usage")</code>	<code>g18/demo.py</code>
seaborn	calling <code>tight_layout</code> on a grid object	<code>jg.fig.tight_layout()</code>	<code>g18/demo.py</code>
seaborn	importing the module	<code>import seaborn as sns</code>	<code>g18/demo.py</code>
seaborn	joint distribution hex plot	<code>jg = sns.jointplot(data=bymo,x=1,y=7,kind='hex')</code>	<code>g18/demo.py</code>
seaborn	setting axis titles on a grid object	<code>jg.set_axis_labels('January','July')</code>	<code>g18/demo.py</code>
seaborn	setting the theme	<code>sns.set_theme(style="white")</code>	<code>g18/demo.py</code>
seaborn	split violin plot	<code>sns.violinplot(data=eights,x="hour",y="usage",hue="mont. . .</code>	<code>g18/demo.py</code>
zipfile	creating a ZipFile object	<code>zip_object = zipfile.ZipFile(zipname)</code>	<code>g11/demo.py</code>
zipfile	importing module	<code>import zipfile</code>	<code>g11/demo.py</code>
zipfile	opening a file in a zip in bytes mode	<code>inp_byte = zip_object.open(csvname)</code>	<code>g11/demo.py</code>