| Module | Description | Example | Script |
|---|---|---|---|
| core | continue, going on to next loop item | continue | g06/demo.py |
| core | dictionary, adding a new entry | co['po'] = 'CO' | g05/demo.py |
| core | dictionary, creating | co = {'name':'Colorado', 'capital':'Denver'} | g05/demo.py |
| core | dictionary, creating via comprehension | word_lengths = { w:len(w) for w in wordlist } | g06/demo.py |
| core | dictionary, iterating through key-value pairs | for w,l in word_lengths.items(): | g06/demo.py |
| core | dictionary, looking up a value | name = ny['name'] | g05/demo.py |
| core | dictionary, making a list of | list1 = [co,ny] | g05/demo.py |
| core | dictionary, obtaining a list of keys | names = super_dict.keys() | g05/demo.py |
| core | f-string, grouping with commas | print(f'Total population: {tot_pop:,}') | g12/demo.py |
| core | f-string, using a formatting string | print( f"PV of {payment} with T={year} and r={r} is ${p... | g08/demo.py |
| core | file, closing | fh.close() | g02/demo.py |
| core | file, opening for reading | fh = open('states.csv') | g05/demo.py |
| core | file, opening for writing | fh = open(filename,"w") | g02/demo.py |
| core | file, output using print | print("It was written during",year,file=fh) | g02/demo.py |
| core | file, output using write | fh.write("Where was this file was written?\n") | g02/demo.py |
| core | file, print without adding spaces | print( '\nOuter:\n', join_o['_merge'].value_counts(), s... | g15/demo.py |
| core | file, reading one line at a time | for line in fh: | g05/demo.py |
| core | for, looping through a list | for n in a_list: | g04/demo.py |
| core | for, looping through a list of tuples | for number,name in div_info: | g14/demo.py |
| core | function, calling | d1_ssq = sumsq(d1) | g07/demo.py |
| core | function, calling with an optional argument | sample_function( 100, 10, r=0.07 ) | g08/demo.py |
| core | function, defining | def sumsq(values: list) -> float: | g07/demo.py |
| core | function, defining with optional argument | def sample_function(payment:float,year:int,r:float=0.05... | g08/demo.py |
| core | function, returning a result | return values | g07/demo.py |
| core | function, using type hinting | def readlist(filename: str) -> list: | g07/demo.py |
| core | if, starting a conditional block | if l == 5: | g06/demo.py |
| core | if, using an elif statement | elif s.isalpha(): | g06/demo.py |
| core | if, using an else statement | else: | g06/demo.py |
| core | list, appending an element | a_list.append("four") | g03/demo.py |
| core | list, create via comprehension | cubes = [n**3 for n in a_list] | g04/demo.py |

| Module | Description | Example | Script |
|---|---|---|---|
| core | list, creating | a_list = [“zero”,“one”,“two”,“three”] | g03/demo.py |
| core | list, determining length | n = len(b_list) | g03/demo.py |
| core | list, extending with another list | a_list.extend(a_more) | g03/demo.py |
| core | list, generating a sequence | b_list = range(1,6) | g04/demo.py |
| core | list, joining with spaces | a_string = ” “.join(a_list) | g03/demo.py |
| core | list, selecting an element | print(a_list[0]) | g03/demo.py |
| core | list, selecting elements 0 to 3 | print(a_list[:4]) | g03/demo.py |
| core | list, selecting elements 1 to 2 | print(a_list[1:3]) | g03/demo.py |
| core | list, selecting elements 1 to the end | print(a_list[1:]) | g03/demo.py |
| core | list, selecting last 3 elements | print(a_list[-3:]) | g03/demo.py |
| core | list, selecting the last element | print(a_list[-1]) | g03/demo.py |
| core | list, sorting | c_sort = sorted(b_list) | g03/demo.py |
| core | list, summing | total = sum(numbers) | g06/demo.py |
| core | math, raising a number to a power | a_cubes.append( n**3 ) | g04/demo.py |
| core | math, rounding a number | rounded = round(ratio,2) | g05/demo.py |
| core | sets, computing difference | print( name_states - pop_states ) | g14/demo.py |
| core | sets, creating | name_states = set( name_data[‘State’] ) | g14/demo.py |
| core | sets, of tuples | tset1 = set( [ (1,2), (2,3), (1,3), (2,3) ] ) | g14/demo.py |
| core | string, concatenating | name = s1+” “+s2+” “+s3 | g02/demo.py |
| core | string, convert to lower case | lower = [s.lower() for s in wordlist] | g06/demo.py |
| core | string, convert to title case | new_s = s.title() | g06/demo.py |
| core | string, converting to an int | value = int(s) | g06/demo.py |
| core | string, creating | filename = “demo.txt” | g02/demo.py |
| core | string, finding starting index | mm_start = long_string.find(“mm”) | g06/demo.py |
| core | string, including a newline character | fh.write(name+“!\n”) | g02/demo.py |
| core | string, is entirely numeric | if s.isnumeric(): | g06/demo.py |
| core | string, matching a substring | has_ñ = [s for s in lower if “ñ” in s] | g06/demo.py |
| core | string, matching end | a_end = [s for s in lower if s.endswith(“a”)] | g06/demo.py |
| core | string, matching multiple starts | ab_start = [s for s in lower if s.startswith(starters)] | g06/demo.py |
| core | string, matching start | a_start = [s for s in lower if s.startswith(“a”)] | g06/demo.py |
| core | string, replacing a substring | words = s.replace(“,”,” “).split() | g06/demo.py |
| core | string, splitting on a comma | parts = line.split(‘,’) | g05/demo.py |
| core | string, splitting on whitespace | b_list = b_string.split() | g03/demo.py |
| core | string, stripping blank space | clean = [item.strip() for item in parts] | g05/demo.py |

| Module | Description | Example | Script |
|---|---|---|---|
| core | tuple, creating | starters = (“a”,“b”,“0”) | g06/demo.py |
| core | type, obtaining for a variable | print( ‘\nraw_states is a DataFrame object:’, type(raw_... | g10/demo.py |
| csv | setting up a DictReader object | reader = csv.DictReader(fh) | g09/demo.py |
| geopandas | drawing a heatmap | near_wv.plot(“mil”,cmap=‘Blues’,legend=True,ax=ax) | g23/demo.py |
| geopandas | extracting geometry from a geodataframe | wv_geo = wv[‘geometry’] | g23/demo.py |
| geopandas | importing the module | import geopandas as gpd | g22/demo.py |
| geopandas | merging data onto a geodataframe | conus = conus.merge(trim,on=‘STATEFP’,how=‘left’,valida... | g23/demo.py |
| geopandas | obtaining coordinates | print( ‘Number of points:’, len(wv_geo.exterior.coords)... | g23/demo.py |
| geopandas | plot with categorical coloring | sel.plot(‘NAME’,cmap=‘Dark2’,ax=ax1) | g23/demo.py |
| geopandas | plotting a boundary | syr.boundary.plot(color=‘gray’,linewidth=1,ax=ax1) | g22/demo.py |
| geopandas | reading a file | syr = gpd.read_file(“tl_2016_36_place-syracuse.zip”) | g22/demo.py |
| geopandas | reading a shapefile | states = gpd.read_file(“cb_2019_us_state_500k.zip”) | g23/demo.py |
| geopandas | testing if rows touch a geometry | touches_wv = conus.touches(wv_geo) | g23/demo.py |
| geopandas | writing a layer to a geodatabase | conus.to_file(“conus.gpkg”,layer=“states”) | g23/demo.py |
| json | importing the module | import json | g05/demo.py |
| json | using to print an object nicely | print( json.dumps(list1,indent=4) ) | g05/demo.py |
| matplotlib | axes, adding a horizontal line | ax21.axhline(medians[‘etr’], c=‘r’, ls=‘–’, lw=1) | g13/demo.py |
| matplotlib | axes, adding a vertical line | ax21.axvline(medians[‘inc’], c=‘r’, ls=‘–’, lw=1) | g13/demo.py |
| matplotlib | axes, labeling the X axis | ax2.set_xlabel(‘Millions’) | g12/demo.py |
| matplotlib | axes, labeling the Y axis | ax1.set_ylabel(‘Millions’) | g12/demo.py |
| matplotlib | axes, turning off a label | ax.set_ylabel(None) | g14/demo.py |
| matplotlib | colors, xkcd palette | syr.plot(color=‘xkcd:lightblue’,ax=ax1) | g22/demo.py |
| matplotlib | figure, adding a title | fig2.suptitle(‘Pooled Data’) | g13/demo.py |
| matplotlib | figure, four panel grid | fig3, axs = plt.subplots(2,2,sharex=True,sharey=True) | g13/demo.py |
| matplotlib | figure, left and right panels | fig2, (ax21,ax22) = plt.subplots(1,2) | g13/demo.py |
| matplotlib | figure, saving | fig2.savefig(‘figure.png’) | g12/demo.py |
| matplotlib | figure, setting the size | fig, axs = plt.subplots(1,2,figsize=(12,6)) | g21/demo.py |
| matplotlib | figure, tuning the layout | fig2.tight_layout() | g12/demo.py |
| matplotlib | figure, working with a list of axes | for ax in axs: | g21/demo.py |
| matplotlib | importing pyplot | import matplotlib.pyplot as plt | g12/demo.py |
| matplotlib | setting the default resolution | plt.rcParams[‘figure.dpi’] = 300 | g12/demo.py |

| Module | Description | Example | Script |
|---|---|---|---|
| matplotlib | using subplots to set up a figure | fig1, ax1 = plt.subplots() | g12/demo.py |
| pandas | RE, replacing a digit or space | unit_part = values.str.replace(r'\d\|\s','',regex=True) | g24/demo.py |
| pandas | RE, replacing a non-digit or space | value_part = values.str.replace(r'\D\|\s','',regex=True) | g24/demo.py |
| pandas | RE, replacing a non-word character | units = units.str.replace(r'\W','',regex=True) | g24/demo.py |
| pandas | columns, dividing along index | by_day_pct = 100*by_day_use.div(by_day_tot,axis='index'... | g18/demo.py |
| pandas | columns, dividing with explicit alignment | normed2 = 100*states.div(pa_row,axis='columns') | g10/demo.py |
| pandas | columns, listing names | print( '\nColumns:', list(raw_states.columns) ) | g10/demo.py |
| pandas | columns, renaming | county = county.rename(columns={'B01001_001E':'pop'}) | g11/demo.py |
| pandas | columns, retrieving one by name | pop = states['pop'] | g10/demo.py |
| pandas | columns, retrieving several by name | print( pop[some_states]/1e6 ) | g10/demo.py |
| pandas | dataframe, appending | gen_all = pd.concat( [gen_oswego, gen_onondaga] ) | g16/demo.py |
| pandas | dataframe, boolean row selection | print( trim[ has_AM ], "\n" ) | g13/demo.py |
| pandas | dataframe, dropping a column | both = both.drop(columns='_merge') | g16/demo.py |
| pandas | dataframe, dropping duplicates | flood = flood.drop_duplicates( subset='TAX_ID' ) | g15/demo.py |
| pandas | dataframe, dropping missing data | merged = geocodes.dropna() | g12/demo.py |
| pandas | dataframe, finding duplicate records | dups = parcels.duplicated( subset='TAX_ID', keep=False... | g15/demo.py |
| pandas | dataframe, getting a block of rows via index | sel = merged.loc[number] | g14/demo.py |
| pandas | dataframe, inner 1:1 merge | join_i = parcels.merge(flood, how='inner', on="TAX_ID",... | g15/demo.py |
| pandas | dataframe, inner join | merged = name_data.merge(pop_data,left_on="State",right... | g14/demo.py |
| pandas | dataframe, left 1:1 merge | join_l = parcels.merge(flood, how='left', on="TAX_ID",... | g15/demo.py |
| pandas | dataframe, left m:1 merge | both = gen_all.merge(plants, how='left', on='Plant Code... | g16/demo.py |
| pandas | dataframe, making a copy | trim = trim.copy() | g13/demo.py |
| pandas | dataframe, melting | long_form = means.reset_index().melt(id_vars='month') | g18/demo.py |
| pandas | dataframe, outer 1:1 merge | join_o = parcels.merge(flood, how='outer', on="TAX_ID",... | g15/demo.py |
| pandas | dataframe, pivoting | by_day_use = usage.pivot(index=['month','day'],columns=... | g18/demo.py |
| pandas | dataframe, reading zipped pickle format | sample2 = pd.read_pickle('sample_pkl.zip') | g17/demo.py |
| pandas | dataframe, resetting the index | hourly = hourly.reset_index() | g18/demo.py |
| pandas | dataframe, right 1:1 merge | join_r = parcels.merge(flood, how='right', on="TAX_ID",... | g15/demo.py |
| pandas | dataframe, saving in zipped pickle format | sample.to_pickle('sample_pkl.zip') | g17/demo.py |
| pandas | dataframe, selecting rows by list indexing | print( low_to_high[ -5: ] ) | g10/demo.py |
| pandas | dataframe, selecting rows via boolean | dup_rec = flood[ dups ] | g15/demo.py |
| pandas | dataframe, selecting rows via query | trimmed = county.query("state =='04' or state == '36' ") | g11/demo.py |
| pandas | dataframe, selective drop of missing data | trim = demo.dropna(subset="Days") | g13/demo.py |
| pandas | dataframe, set index keeping the column | states = states.set_index('STUSPS',drop=False) | g23/demo.py |

| Module | Description | Example | Script |
|---|---|---|---|
| pandas | dataframe, shape attribute | print( 'number of rows, columns:', conus.shape ) | g23/demo.py |
| pandas | dataframe, sorting by a column | county = county.sort_values('pop') | g11/demo.py |
| pandas | dataframe, sorting by index | summary = summary.sort_index(ascending=False) | g16/demo.py |
| pandas | dataframe, summing a boolean | print( '\nduplicate parcels:', dups.sum() ) | g15/demo.py |
| pandas | dataframe, summing across columns | by_day_tot = by_day_use.sum(axis='columns') | g18/demo.py |
| pandas | dataframe, unstacking an index level | bymo = bymo.unstack('month') | g18/demo.py |
| pandas | dataframe, using a multilevel column index | means = grid['mean'] | g21/demo.py |
| pandas | dataframe, using xs to select a subset | print( county.xs('04',level='state') ) | g11/demo.py |
| pandas | dataframe, using xs with columns | c1 = grid.xs('c1',axis='columns',level=1) | g21/demo.py |
| pandas | dataframe, writing to a CSV file | merged.to_csv('demo-merged.csv') | g14/demo.py |
| pandas | datetime, building via to_datetime() | date = pd.to_datetime(recs['ts']) | g15/demo.py |
| pandas | datetime, building with a format | ymd = pd.to_datetime( sample['TRANSACTION_DT'], format=... | g17/demo.py |
| pandas | datetime, extracting day attribute | recs['day'] = date.dt.day | g15/demo.py |
| pandas | datetime, extracting hour attribute | recs['hour'] = date.dt.hour | g15/demo.py |
| pandas | general, display information about object | sample.info() | g17/demo.py |
| pandas | general, displaying all columns | pd.set_option('display.max_columns',None) | g17/demo.py |
| pandas | general, displaying all rows | pd.set_option('display.max_rows', None) | g10/demo.py |
| pandas | general, importing the module | import pandas as pd | g10/demo.py |
| pandas | general, using copy_on_write mode | pd.options.mode.copy_on_write = True | g17/demo.py |
| pandas | general, using qcut to create deciles | dec = pd.qcut( county['pop'], 10, labels=range(1,11) ) | g11/demo.py |
| pandas | groupby, cumulative sum within group | cumulative_inc = group_by_state['pop'].cumsum() | g11/demo.py |
| pandas | groupby, descriptive statistics | inc_stats = group_by_state['pop'].describe() | g11/demo.py |
| pandas | groupby, iterating over groups | for t,g in group_by_state: | g11/demo.py |
| pandas | groupby, median of each group | pop_med = group_by_state['pop'].median() | g11/demo.py |
| pandas | groupby, quantile of each group | pop_25th = group_by_state['pop'].quantile(0.25) | g11/demo.py |
| pandas | groupby, return group number | groups = group_by_state.ngroup() | g11/demo.py |
| pandas | groupby, return number within group | seqnum = group_by_state.cumcount() | g11/demo.py |
| pandas | groupby, return rank within group | rank_age = group_by_state['pop'].rank() | g11/demo.py |
| pandas | groupby, select first records | first2 = group_by_state.head(2) | g11/demo.py |
| pandas | groupby, select largest values | largest = group_by_state['pop'].nlargest(2) | g11/demo.py |
| pandas | groupby, select last records | last2 = group_by_state.tail(2) | g11/demo.py |
| pandas | groupby, size of each group | num_rows = group_by_state.size() | g11/demo.py |
| pandas | groupby, sum of each group | state = county.groupby('state')['pop'].sum() | g11/demo.py |

| Module | Description | Example | Script |
|---|---|---|---|
| pandas | index, creating with 3 levels | county = county.set_index(['state','county', 'NAME']) | g11/demo.py |
| pandas | index, listing names | print( '\nIndex (rows):', list(raw_states.index) ) | g10/demo.py |
| pandas | index, renaming values | div_pop = div_pop.rename(index=div_names) | g12/demo.py |
| pandas | index, retrieving a row by name | pa_row = states.loc['Pennsylvania'] | g10/demo.py |
| pandas | index, retrieving first rows by location | print( low_to_high.iloc[ 0:10 ] ) | g10/demo.py |
| pandas | index, retrieving last rows by location | print( low_to_high.iloc[ -5: ] ) | g10/demo.py |
| pandas | index, setting to a column | states = raw_states.set_index('name') | g10/demo.py |
| pandas | plotting, bar plot | reg_pop.plot.bar(title='Population',ax=ax1) | g12/demo.py |
| pandas | plotting, histogram | hh_data['etr'].plot.hist(ax=ax1,bins=20,title='Distribu... | g13/demo.py |
| pandas | plotting, horizontal bar plot | div_pop.plot.barh(title='Population',ax=ax2) | g12/demo.py |
| pandas | plotting, scatter colored by 3rd var | tidy_data.plot.scatter(ax=ax4,x='Income',y='ETR',c='typ... | g13/demo.py |
| pandas | plotting, scatter plot | hh_data.plot.scatter(ax=ax21,x='inc',y='etr',title='ETR... | g13/demo.py |
| pandas | plotting, turning off legend | sel.plot.barh(x='Name',y='percent',ax=ax,legend=None) | g14/demo.py |
| pandas | reading, csv data | raw_states = pd.read_csv('state-data.csv') | g10/demo.py |
| pandas | reading, from an open file handle | gen_oswego = pd.read_csv(fh1) | g16/demo.py |
| pandas | reading, setting index column | state_data = pd.read_csv('state-data.csv',index_col='na... | g12/demo.py |
| pandas | reading, using dtype dictionary | county = pd.read_csv('county_pop.csv',dtype=fips) | g11/demo.py |
| pandas | series, RE at start | is_LD = trim['Number'].str.contains(r"1\|2") | g13/demo.py |
| pandas | series, applying a function to each element | name_clean = name_parts.apply(' '.join) | g24/demo.py |
| pandas | series, automatic alignment by index | merged['percent'] = 100*merged['pop']/div_pop | g14/demo.py |
| pandas | series, combining via where() | mod['comb_units'] = unit_part.where( unit_part!='' , mo... | g24/demo.py |
| pandas | series, contains RE or RE | is_TT = trim['Days'].str.contains(r"Tu\|Th") | g13/demo.py |
| pandas | series, contains a plain string | has_AM = trim['Time'].str.contains("AM") | g13/demo.py |
| pandas | series, contains an RE | has_AMPM = trim['Time'].str.contains("AM.*PM") | g13/demo.py |
| pandas | series, converting strings to title case | fixname = subset_view['NAME'].str.title() | g17/demo.py |
| pandas | series, converting to a list | print( name_data['State'].to_list() ) | g14/demo.py |
| pandas | series, converting to lower case | name = mod['name'].str.lower() | g24/demo.py |
| pandas | series, dropping rows using a list | conus = states.drop(not_conus) | g23/demo.py |
| pandas | series, element-by-element or | is_either = is_ca \| is_tx | g17/demo.py |
| pandas | series, filling missing values | mod['comb_units'] = mod['comb_units'].fillna('feet') | g24/demo.py |
| pandas | series, removing spaces | units = units.str.strip() | g24/demo.py |
| pandas | series, replacing values using a dictionary | units = units.replace(spellout) | g24/demo.py |
| pandas | series, retrieving an element | print( "\nFlorida's population:", pop['Florida']/1e6 ) | g10/demo.py |
| pandas | series, sort in decending order | div_pop = div_pop.sort_values(ascending=False) | g12/demo.py |

| Module | Description | Example | Script |
|---|---|---|---|
| pandas | series, sorting by value | low_to_high = normed['med_pers_inc'].sort_values() | g10/demo.py |
| pandas | series, splitting strings on whitespace | name_parts = name.str.split() | g24/demo.py |
| pandas | series, splitting via RE | trim['Split'] = trim["Time"].str.split(r":\| - \| ") | g13/demo.py |
| pandas | series, splitting with expand | exp = trim["Time"].str.split(r":\| - \| ", expand=True) | g13/demo.py |
| pandas | series, summing | reg_pop = by_reg['pop'].sum()/1e6 | g12/demo.py |
| pandas | series, unstacking | tot_wide = tot_amt.unstack('PGI') | g17/demo.py |
| pandas | series, using isin() | fixed = flood['TAX_ID'].isin( dup_rec['TAX_ID'] ) | g15/demo.py |
| pandas | series, using value_counts() | print( '\nOuter:\n', join_o['_merge'].value_counts(), s... | g15/demo.py |
| requests | calling the get() method | response = requests.get(api,payload) | g19/demo.py |
| requests | checking the URL | print( 'url:', response.url ) | g19/demo.py |
| requests | checking the response text | print( response.text ) | g19/demo.py |
| requests | checking the status code | print( 'status:', response.status_code ) | g19/demo.py |
| requests | decoding a JSON response | rows = response.json() | g19/demo.py |
| requests | importing the module | import requests | g19/demo.py |
| scipy | calling newton's method | cr = opt.newton(find_cube_root,xinit,maxiter=20,args=[y... | g08/demo.py |
| scipy | importing the module | import scipy.optimize as opt | g08/demo.py |
| seaborn | adding a title to a grid object | jg.fig.suptitle('Distribution of Hourly Load') | g18/demo.py |
| seaborn | barplot | hue='month',palette='deep',ax=ax1) | g18/demo.py |
| seaborn | basic violin plot | sns.violinplot(data=janjul,x="month",y="usage") | g18/demo.py |
| seaborn | boxenplot | sns.boxenplot(data=janjul,x="month",y="usage") | g18/demo.py |
| seaborn | calling tight_layout on a grid object | jg.fig.tight_layout() | g18/demo.py |
| seaborn | drawing a heatmapped grid | sns.heatmap(means,annot=True,fmt=".0f",cmap='Spectral',... | g21/demo.py |
| seaborn | importing the module | import seaborn as sns | g18/demo.py |
| seaborn | joint distribution hex plot | jg = sns.jointplot(data=bymo,x=1,y=7,kind='hex') | g18/demo.py |
| seaborn | line plot | sns.lineplot(data=long_form,x='hour',y='value',hue='mon... | g18/demo.py |
| seaborn | setting axis titles on a grid object | jg.set_axis_labels('January','July') | g18/demo.py |
| seaborn | setting the theme | sns.set_theme(style="white") | g18/demo.py |
| seaborn | split violin plot | hue="month",palette='deep',split=True) | g18/demo.py |
| zipfile | importing the module | import zipfile | g16/demo.py |
| zipfile | opening a file in an archive | fh1 = archive.open('generators-oswego.csv') | g16/demo.py |
| zipfile | opening an archive | archive = zipfile.ZipFile('generators.zip') | g16/demo.py |
| zipfile | reading the list of files | print( archive.namelist() ) | g16/demo.py |