

| Module | Description                                 | Example  | Script      |
|--------|---|--|-------------|
| core   | dictionary, adding a new entry              | <code>co['po'] = 'CO'</code>   | g05/demo.py |
| core   | dictionary, creating                        | <code>co = {'name':'Colorado', 'capital':'Denver'}</code>                  | g05/demo.py |
| core   | dictionary, creating via comprehension      | <code>fips_cols = {col:str for col in fips_vars}</code>                    | g13/demo.py |
| core   | dictionary, looking up a value              | <code>name = ny['name']</code>   | g05/demo.py |
| core   | dictionary, making a list of                | <code>list1 = [co,ny]</code>   | g05/demo.py |
| core   | dictionary, obtaining a list of keys        | <code>names = super_dict.keys()</code>                                     | g05/demo.py |
| core   | f-string, using a formatting string         | <code>print( f"PV of {payment} with T={year} and r={r} is \${p. . .</code> | g07/demo.py |
| core   | file, closing                               | <code>fh.close()</code>  | g02/demo.py |
| core   | file, opening for reading                   | <code>fh = open('states.csv')</code>                                       | g05/demo.py |
| core   | file, opening for writing                   | <code>fh = open(filename,"w")</code>                                       | g02/demo.py |
| core   | file, output using print                    | <code>print("It was written during",year,file=fh)</code>                   | g02/demo.py |
| core   | file, output using write                    | <code>fh.write("Where was this file was written?\n")</code>                | g02/demo.py |
| core   | file, reading one line at a time            | <code>for line in fh:</code>   | g05/demo.py |
| core   | for, looping through a list                 | <code>for n in a_list:</code>  | g04/demo.py |
| core   | for, looping through a list of tuples       | <code>for number,name in div_info:</code>                                  | g13/demo.py |
| core   | function, calling                           | <code>d1_ssq = sumsq(d1)</code>  | g06/demo.py |
| core   | function, calling with an optional argument | <code>sample_function( 100, 10, r=0.07 )</code>                            | g07/demo.py |
| core   | function, defining                          | <code>def sumsq(values):</code>  | g06/demo.py |
| core   | function, defining with optional argument   | <code>def sample_function(payment,year,r=0.05):</code>                     | g07/demo.py |
| core   | function, returning a result                | <code>return values</code>   | g06/demo.py |
| core   | list, appending an element                  | <code>a_list.append("four")</code>   | g03/demo.py |
| core   | list, create via comprehension              | <code>cubes = [n**3 for n in a_list]</code>                                | g04/demo.py |
| core   | list, creating                              | <code>a_list = ["zero","one","two","three"]</code>                         | g03/demo.py |
| 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 |

| Module     | Description                           | Example  | Script      |
|------------|---------------------------------------|--|-------------|
| 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       | sets, computing difference            | <code>print( name_states - pop_states )</code>                             | g13/demo.py |
| core       | sets, creating                        | <code>name_states = set( name_data['State'] )</code>                       | g13/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, 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       | type, obtaining for a variable        | <code>print( '\nraw_states is a DataFrame object:', type(raw_ . . .</code> | g09/demo.py |
| csv        | setting up a DictReader object        | <code>reader = csv.DictReader(fh)</code>                                   | g08/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, adding a horizontal line        | <code>ax21.axhline(medians['etr'], c='r', ls='-', lw=1)</code>             | g12/demo.py |
| matplotlib | axes, adding a vertical line          | <code>ax21.axvline(medians['inc'], c='r', ls='-', lw=1)</code>             | g12/demo.py |
| matplotlib | axes, labeling the X axis             | <code>ax1.set_xlabel('Millions')</code>                                    | g11/demo.py |
| matplotlib | axes, labeling the Y axis             | <code>ax1.set_ylabel("Population, Millions")</code>                        | g11/demo.py |
| matplotlib | axes, setting a title                 | <code>ax1.set_title('Population')</code>                                   | g11/demo.py |
| matplotlib | axes, turning off the label           | <code>ax.set_ylabel(None)</code>   | g13/demo.py |
| matplotlib | figure, adding a title                | <code>fig2.suptitle('Pooled Data')</code>                                  | g12/demo.py |
| matplotlib | figure, four panel grid               | <code>fig3, axs = plt.subplots(2,2,sharex=True,sharey=True)</code>         | g12/demo.py |
| matplotlib | figure, left and right panels         | <code>fig2, (ax21,ax22) = plt.subplots(1,2)</code>                         | g12/demo.py |
| matplotlib | figure, saving                        | <code>fig1.savefig('figure.png')</code>                                    | g11/demo.py |
| matplotlib | figure, tuning the layout             | <code>fig1.tight_layout()</code>   | g11/demo.py |
| matplotlib | importing pyplot                      | <code>import matplotlib.pyplot as plt</code>                               | g11/demo.py |

| Module     | Description                                  | Example   | Script                   |
|------------|--|---|--------------------------|
| matplotlib | setting the default resolution               | <code>plt.rcParams['figure.dpi'] = 300</code>                             | <code>g11/demo.py</code> |
| matplotlib | using subplots to set up a figure            | <code>fig1, ax1 = plt.subplots()</code>                                   | <code>g11/demo.py</code> |
| pandas     | columns, dividing with explicit alignment    | <code>normed2 = 100*states.div(pa_row,axis='columns')</code>              | <code>g09/demo.py</code> |
| pandas     | columns, listing names                       | <code>print( '\nColumns:', list(raw_states.columns) )</code>              | <code>g09/demo.py</code> |
| pandas     | columns, renaming                            | <code>county = county.rename(columns={'B01001_001E':'pop'})</code>        | <code>g10/demo.py</code> |
| pandas     | columns, retrieving one by name              | <code>pop = states['pop']</code>  | <code>g09/demo.py</code> |
| pandas     | columns, retrieving several by name          | <code>print( pop[some_states]/1e6 )</code>                                | <code>g09/demo.py</code> |
| pandas     | dataframe, boolean row selection             | <code>print( trim[ has_AM ], "\n" )</code>                                | <code>g12/demo.py</code> |
| pandas     | dataframe, dropping missing data             | <code>trim = demo.dropna(subset="Days")</code>                            | <code>g12/demo.py</code> |
| pandas     | dataframe, getting a block of rows via index | <code>sel = merged.loc[number]</code>                                     | <code>g13/demo.py</code> |
| pandas     | dataframe, inner join                        | <code>merged = name_data.merge(pop_data,left_on="State",right. . .</code> | <code>g13/demo.py</code> |
| pandas     | dataframe, making a copy                     | <code>trim = trim.copy()</code>   | <code>g12/demo.py</code> |
| pandas     | dataframe, selecting rows by list indexing   | <code>print( low_to_high[ -5: ] )</code>                                  | <code>g09/demo.py</code> |
| pandas     | dataframe, selecting rows via query          | <code>trimmed = county.query("state == '04' or state == '36' ")</code>    | <code>g10/demo.py</code> |
| pandas     | dataframe, sorting by a column               | <code>county = county.sort_values('pop')</code>                           | <code>g10/demo.py</code> |
| pandas     | dataframe, using xs to select a subset       | <code>print( county.xs('04',level='state') )</code>                       | <code>g10/demo.py</code> |
| pandas     | dataframe, writing to a CSV file             | <code>merged.to_csv('demo-merged.csv')</code>                             | <code>g13/demo.py</code> |
| pandas     | general, displaying all rows                 | <code>pd.set_option('display.max_rows', None)</code>                      | <code>g09/demo.py</code> |
| pandas     | general, importing the module                | <code>import pandas as pd</code>  | <code>g09/demo.py</code> |
| pandas     | general, using qcut to create deciles        | <code>dec = pd.qcut( county['pop'], 10, labels=range(1,11) )</code>       | <code>g10/demo.py</code> |
| pandas     | groupby, cumulative sum within group         | <code>cumulative_inc = group_by_state['pop'].cumsum()</code>              | <code>g10/demo.py</code> |
| pandas     | groupby, descriptive statistics              | <code>inc_stats = group_by_state['pop'].describe()</code>                 | <code>g10/demo.py</code> |
| pandas     | groupby, iterating over groups               | <code>for t,g in group_by_state:</code>                                   | <code>g10/demo.py</code> |
| pandas     | groupby, median of each group                | <code>pop_med = group_by_state['pop'].median()</code>                     | <code>g10/demo.py</code> |
| pandas     | groupby, quantile of each group              | <code>pop_25th = group_by_state['pop'].quantile(0.25)</code>              | <code>g10/demo.py</code> |
| pandas     | groupby, return group number                 | <code>groups = group_by_state.ngroup()</code>                             | <code>g10/demo.py</code> |
| pandas     | groupby, return number within group          | <code>seqnum = group_by_state.cumcount()</code>                           | <code>g10/demo.py</code> |
| pandas     | groupby, return rank within group            | <code>rank_age = group_by_state['pop'].rank()</code>                      | <code>g10/demo.py</code> |
| pandas     | groupby, select first records                | <code>first2 = group_by_state.head(2)</code>                              | <code>g10/demo.py</code> |
| pandas     | groupby, select largest values               | <code>largest = group_by_state['pop'].nlargest(2)</code>                  | <code>g10/demo.py</code> |
| pandas     | groupby, select last records                 | <code>last2 = group_by_state.tail(2)</code>                               | <code>g10/demo.py</code> |
| pandas     | groupby, size of each group                  | <code>num_rows = group_by_state.size()</code>                             | <code>g10/demo.py</code> |
| pandas     | groupby, sum of each group                   | <code>state = county.groupby('state')['pop'].sum()</code>                 | <code>g10/demo.py</code> |

| Module | Description                              | Example   | Script      |
|--------|--|---|-------------|
| pandas | index, creating with 3 levels            | <code>county = county.set_index(['state','county', 'NAME'])</code>        | g10/demo.py |
| pandas | index, listing names                     | <code>print( '\nIndex (rows):', list(raw_states.index) )</code>           | g09/demo.py |
| pandas | index, renaming values                   | <code>div_pop = div_pop.rename(index=div_names)</code>                    | g11/demo.py |
| pandas | index, retrieving a row by name          | <code>pa_row = states.loc['Pennsylvania']</code>                          | g09/demo.py |
| pandas | index, retrieving first rows by location | <code>print( low_to_high.iloc[ 0:10 ] )</code>                            | g09/demo.py |
| pandas | index, retrieving last rows by location  | <code>print( low_to_high.iloc[ -5: ] )</code>                             | g09/demo.py |
| pandas | index, setting to a column               | <code>states = raw_states.set_index('name')</code>                        | g09/demo.py |
| pandas | plotting, bar plot                       | <code>reg_pop.plot.bar(ax=ax1)</code>                                     | g11/demo.py |
| pandas | plotting, histogram                      | <code>hh_data['etr'].plot.hist(ax=ax0,bins=20,title='Distribu. . .</code> | g12/demo.py |
| pandas | plotting, horizontal bar plot            | <code>div_pop.plot.barh(ax=ax1)</code>                                    | g11/demo.py |
| pandas | plotting, scatter colored by 3rd var     | <code>tidy_data.plot.scatter(ax=ax4,x='Income',y='ETR',c='typ. . .</code> | g12/demo.py |
| pandas | plotting, scatter plot                   | <code>hh_data.plot.scatter(ax=ax21,x='inc',y='etr',title='ETR. . .</code> | g12/demo.py |
| pandas | plotting, turning off legend             | <code>sel.plot.barh(x='Name',y='percent',ax=ax,legend=None)</code>        | g13/demo.py |
| pandas | reading, csv data                        | <code>raw_states = pd.read_csv('state-data.csv')</code>                   | g09/demo.py |
| pandas | reading, setting index column            | <code>state_data = pd.read_csv('state-data.csv',index_col='na. . .</code> | g11/demo.py |
| pandas | reading, using dtype dictionary          | <code>county = pd.read_csv('county_pop.csv',dtype=fips)</code>            | g10/demo.py |
| pandas | series, RE at start                      | <code>is_LD = trim['Number'].str.contains(r"1 2")</code>                  | g12/demo.py |
| pandas | series, automatic alignment by index     | <code>merged['percent'] = 100*merged['pop']/div_pop</code>                | g13/demo.py |
| pandas | series, contains RE or RE                | <code>is_TT = trim['Days'].str.contains(r"Tu Th")</code>                  | g12/demo.py |
| pandas | series, contains a plain string          | <code>has_AM = trim['Time'].str.contains("AM")</code>                     | g12/demo.py |
| pandas | series, contains an RE                   | <code>has_AMPM = trim['Time'].str.contains("AM.*PM")</code>               | g12/demo.py |
| pandas | series, converting to a list             | <code>print( name_data['State'].to_list() )</code>                        | g13/demo.py |
| pandas | series, retrieving an element            | <code>print( "\nFlorida's population:", pop['Florida']/1e6 )</code>       | g09/demo.py |
| pandas | series, sort in decending order          | <code>div_pop = div_pop.sort_values(ascending=False)</code>               | g11/demo.py |
| pandas | series, sorting by value                 | <code>low_to_high = normed['med_pers_inc'].sort_values()</code>           | g09/demo.py |
| pandas | series, splitting via RE                 | <code>trim['Split'] = trim["Time"].str.split(r":  -   ")</code>           | g12/demo.py |
| pandas | series, splitting with expand            | <code>exp = trim["Time"].str.split(r":  -   ", expand=True)</code>        | g12/demo.py |
| pandas | series, summing                          | <code>reg_pop = by_reg['pop'].sum()/1e6</code>                            | g11/demo.py |
| scipy  | calling newton's method                  | <code>cr = opt.newton(find_cube_root,xinit,maxiter=20,args=[y. . .</code> | g07/demo.py |
| scipy  | importing the module                     | <code>import scipy.optimize as opt</code>                                 | g07/demo.py |