| Module | Description | Example | Script |
|---|---|---|---|
| collections | creating a defaultdict of lists | by_zone = defaultdict(list) | g10/demo.py |
| collections | importing defaultdict | from collections import defaultdict | g10/demo.py |
| | | | |
| core | dictionary, adding a new entry | co['po'] = 'CO' | g05/demo.py |
| core | dictionary, checking for existing key | if fips in name_by_fips: | g09/demo.py |
| core | dictionary, creating | co = {'name':'Colorado', 'capital':'Denver'} | g05/demo.py |
| core | dictionary, deleting an entry | del name_by_fips["00"] | g09/demo.py |
| core | dictionary, iterating over keys | for fips in name_by_fips.keys(): | g09/demo.py |
| core | dictionary, iterating over values | for rec in name_by_fips.values(): | g09/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 | dictionary, sorting keys | for tz in sorted( by_zone.keys() ): | g10/demo.py |
| | | | |
| core | f-string, using a formatting string | print( f"PV of {payment} with T={year} and r={r} is ${pv}" ) | g07/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, 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 | function, calling | d1_ssq = sumsq(d1) | g06/demo.py |
| core | function, calling with an optional argument | sample_function( 100, 10, r=0.07 ) | g07/demo.py |
| core | function, defining | def sumsq(values): | g06/demo.py |
| core | function, defining with optional argument | def sample_function(payment,year,r=0.05): | g07/demo.py |
| core | function, returning a result | return values | g06/demo.py |
| | | | |
| core | if statement, testing for equality | if fips == "36": | g09/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 |
| core | list, creating | a_list = ["zero","one","two","three"] | g03/demo.py |
| core | list, determining length | n = len(b_list) | g03/demo.py |

| Module | Description | Example | Script |
|---|---|---|---|
| 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 | tot_inc = sum(incomes) | g08/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 | string, concatenating | name = s1+" "+s2+" "+s3 | g02/demo.py |
| core | string, converting to an int | values.append( int(line) ) | g06/demo.py |
| core | string, converting to title case | name = codes[key].title() | g11/demo.py |
| core | string, creating | filename = "demo.txt" | g02/demo.py |
| core | string, including a newline character | fh.write(name+"!\n") | g02/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 |
| | | | |
| core | tuple, creating | this_tuple = (med_density,state) | g10/demo.py |
| core | tuple, creating via split | (last,first) = name.split(',') | g11/demo.py |
| core | tuple, looping over | for (den,state) in sorted(by_density): | g10/demo.py |
| core | tuple, sorting | for key in sorted(codes): | g11/demo.py |
| core | tuple, testing equality of | if key == (29,'VA'): | g11/demo.py |
| | | | |
| csv | opening a file for use with DictWriter | fh = open(outfile,'w',newline='') | g09/demo.py |
| csv | setting up a DictReader object | reader = csv.DictReader(fh) | g08/demo.py |
| csv | setting up a DictWriter object | writer = csv.DictWriter(fh,fields) | g09/demo.py |
| csv | using DictReader with a list | reader = csv.DictReader(lines) | g10/demo.py |
| csv | writing a header with DictWriter | writer.writeheader() | g09/demo.py |
| csv | writing a record with DictWriter | writer.writerow(name_rec) | g09/demo.py |

| Module | Description | Example | Script |
|---|---|---|---|
| io | converting a byte stream to characters | inp_handle = io.TextIOWrapper(inp_byte) | g11/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 |
| numpy | computing a median | med_density = round( np.median(this_list), 2 ) | g10/demo.py |
| numpy | importing | import numpy as np | g10/demo.py |
| pandas | columns, dividing with explicit alignment | normed2 = 100*states.div(pa_row,axis='columns') | g12/demo.py |
| pandas | columns, listing names | print( '\nColumns:', list(states.columns) ) | g12/demo.py |
| pandas | columns, retrieving one by name | pop = states['pop'] | g12/demo.py |
| pandas | columns, retrieving several by name | print( pop[some_states]/1e6 ) | g12/demo.py |
| pandas | displaying all rows | pd.set_option('display.max_rows', None) | g12/demo.py |
| pandas | importing the module | import pandas as pd | g12/demo.py |
| pandas | index, listing names | print( '\nIndex (rows):', list(states.index) ) | g12/demo.py |
| pandas | index, retrieving a row by name | pa_row = states.loc['Pennsylvania'] | g12/demo.py |
| pandas | index, retrieving first rows by location | print( low_to_high.iloc[ 0:10 ] ) | g12/demo.py |
| pandas | index, retrieving last rows by location | print( low_to_high.iloc[ -5: ] ) | g12/demo.py |
| pandas | index, setting to a column | new_states = states.set_index('name') | g12/demo.py |
| pandas | index, setting to a column in place | states.set_index('name',inplace=True) | g12/demo.py |
| pandas | reading, CSV data | states = pd.read_csv('state-data.csv') | g12/demo.py |
| pandas | series, retrieving an element | print( "\nFlorida's population:", pop['Florida']/1e6 ) | g12/demo.py |
| pandas | series, sorting by value | low_to_high = normed['med_pers_inc'].sort_values() | g12/demo.py |
| scipy | calling newton's method | cr = opt.newton(find_cube_root,xinit,maxiter=20,args=[y]) | g07/demo.py |
| scipy | importing the module | import scipy.optimize as opt | g07/demo.py |
| zipfile | creating a ZipFile object | zip_object = zipfile.ZipFile(zipname) | g11/demo.py |
| zipfile | importing module | import zipfile | g11/demo.py |
| zipfile | opening a file in a zip in bytes mode | inp_byte = zip_object.open(csvname) | g11/demo.py |