

Module	Description	Example	Script
collections	creating a defaultdict of lists	<code>by_zone = defaultdict(list)</code>	<code>g10/demo.py</code>
collections	importing defaultdict	<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 \${pv}")</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, 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_ssq = 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>
core	list, determining length	<code>n = len(b_list)</code>	<code>g03/demo.py</code>

Module	Description	Example	Script
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
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, 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	displaying all rows	<code>pd.set_option('display.max_rows', None)</code>	g12/demo.py
pandas	importing the module	<code>import pandas as pd</code>	g12/demo.py
pandas	index, listing names	<code>print('\nIndex (rows):', list(states.index))</code>	g12/demo.py
pandas	index, retrieving a row by name	<code>pa_row = states.loc['Pennsylvania']</code>	g12/demo.py
pandas	index, retrieving first rows by location	<code>print(low_to_high.iloc[0:10])</code>	g12/demo.py
pandas	index, retrieving last rows by location	<code>print(low_to_high.iloc[-5:])</code>	g12/demo.py
pandas	index, setting to a column	<code>new_states = states.set_index('name')</code>	g12/demo.py
pandas	index, setting to a column in place	<code>states.set_index('name',inplace=True)</code>	g12/demo.py
pandas	reading, CSV data	<code>states = pd.read_csv('state-data.csv')</code>	g12/demo.py
pandas	series, retrieving an element	<code>print("\nFlorida's population:", pop['Florida']/1e6)</code>	g12/demo.py
pandas	series, sorting by value	<code>low_to_high = normed['med_pers_inc'].sort_values()</code>	g12/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
zipfile	creating a ZipFile object	<code>zip_object = zipfile.ZipFile(zipname)</code>	g11/demo.py
zipfile	importing module	<code>import zipfile</code>	g11/demo.py
zipfile	opening a file in a zip in bytes mode	<code>inp_byte = zip_object.open(csvname)</code>	g11/demo.py