core dictionary, creating co = {\text{iname'}:\text{Colorado', 'capital':\text{Denver'}}} \ g05/demo.g05/nato.pr core dictionary, loghing up a value name = my[name] rore dictionary, looping over keys for fips in name_by_fips.keys():	Module	Description	Example	Script
core dictionary, creating co = {\text{iname'}:\text{Colorado', 'capital':\text{Denver'}}} \ g05/demo.g05/nato.pr core dictionary, loghing up a value name = my[name] rore dictionary, looping over keys for fips in name_by_fips.keys():	core	dictionary, adding a new entry	co['po'] = 'CO'	g05/demo.py
core dictionary, looking up a value name = ny[name] g05/demo.; core dictionary, looping over keys for fips in name_by_fips.keys(): g99/demo.; core dictionary, looping over values for rec in name_by_fips.values(): g99/demo.; core dictionary, making a list of list1 = [co.ny] g05/demo.; g	core	dictionary, creating	co = {'name':'Colorado', 'capital':'Denver'}	g05/demo.py
core dictionary, looping over keys for fips in name_by_fips.keys(): g09/demo.gore dictionary, making a list of list1 = [co,ny] g05/demo.gore dictionary, making a list of list1 = [co,ny] g05/demo.gore dictionary, making a list of keys fields = list(name_by_fips["36"].keys()) g09/demo.gore dictionary, obtaining a list of keys names = super_dict.keys() g05/demo.gofo/	core	dictionary, length of	$n = len(to_nato)$	g05/nato.py
core dictionary, looping over values for rec in name_by_fips.values(): core dictionary, making a list of list1 = [co,ny] core dictionary, making a list of keys fields = list(name_by_fips["36"].keys())	core	dictionary, looking up a value	name = ny['name']	g05/demo.py
core dictionary, making a list of core dictionary, making a list of keys fields = list(name_by_fips["36"].keys()) g09/demo_t g05/demo_t g05/d	core	dictionary, looping over keys	for fips in name_by_fips.keys():	g09/demo.py
core dictionary, making a list of keys names = super_dict.keys()	core	dictionary, looping over values	for rec in name_by_fips.values():	g09/demo.py
core dictionary, obtaining a list of keys names = super_dict.keys() core f-string, using a formatting string print (f"PV of {payment} with T={year} and r={r} is \${pv}") g07/demo.gore file, closing fh.close() ffle, opening for reading fh.close() ffle, opening for writing fh = open(filename,"w") core file, output using print print("It was written during",year,file=fh) g02/demo.gore file, output using write fh.write("Where was this file was written?\n") core file, reading one line at a time for line in fh: core function, calling with an optional argument core function, defining with optional argument core function, defining with optional argument core function, returning a result return values core list, appending an element core list, create via comprehension core list, create via comprehension core list, create via comprehension core list, ceterming length n = len(b_list) earth of the strength of the strength of the print of the strength of the print	core	dictionary, making a list of	list1 = [co,ny]	g05/demo.py
core file, closing file, opening for reading file, opening for reading file, opening for rewriting file, opening for rewriting file, opening for rewriting file, output using write file, output using write file, reading one line at a time for line in fh: core function, calling core function, defining with an optional argument core function, defining with optional argument core function, returning a result return values core list, appending an element core list, create via comprehension core list, creating a sequence list, generating a sequence list, ipoining with spaces file, closing file, closing file, close() g02/demo. g05/demo. g02/demo. g02/demo. g02/demo. g02/demo. g06/demo. g	core	dictionary, making a list of keys	fields = list(name_by_fips["36"].keys())	g09/demo.py
core file, closing for reading fh.close() core file, opening for reading fh = open('states.csv') core file, opening for writing fh = open(filename,"w") core file, output using print print("lt was written during",year,file=fh) core file, output using write fh.write("Where was this file was written?\n") core file, reading one line at a time for line in fh: core function, calling core function, calling dunction defining with an optional argument core function, defining with optional argument core function, returning a result core if statement, inequality test if fips != '00': core list, appending an element a_list.append("four") core list, create via comprehension cubes = [n**3 for n in a_list] core list, creating a_list.append("four") core list, creating a_list.append("four") core list, creating a_list.append("me", "three") core list, determining length n = len(b_list) core list, generating a sequence b_list = range(1.6) a_list.append(a_more) b_l	core	dictionary, obtaining a list of keys	names = super_dict.keys()	g05/demo.py
core file, opening for reading fh = open('states.csv') core file, opening for writing fh = open(filename,"w") core file, output using print print("It was written during",year,file=fh) core file, output using write print("Where was this file was written?\n") core file, reading one line at a time for line in fh: core function, calling d1_ssq = sumsq(d1) core function, calling def sumsq(values): core function, defining def sumsq(values): core function, defining with optional argument core function, returning a result return values core if statement, inequality test if fips != '00': core list, appending an element a_list. psend("four") core list, create via comprehension cubes = [n**3 for n in a_list] core list, creating a sequence b_list = range(1,6) core list, gioning with spaces g03/demo.g03/demo.g03/demo.g03/demo.g03/demo.go3	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, opening for writing file open (filename, "w") g02/demo.goze file, output using print print print ("It was written during", year, file=fh) g02/demo.goze file, output using write file, write ("Where was this file was written?\n") g02/demo.goze file, reading one line at a time for line in fh: g05/demo.goze for, looping through a list for n in a_list: g04/demo.goze function, calling d1_ssq = sumsq(d1) sample_function(100, 10, r=0.07) g07/demo.goze function, defining def sample_function(100, 10, r=0.07) g07/demo.goze function, returning a result def sample_function(payment, year, r=0.05): g07/demo.goze function, returning a result return values g06/demo.goze filst, create via comprehension cubes = [n**3 for n in a_list] g03/demo.goze list, creating a_list = ["zero", "one", "two", "three"] g03/demo.goze list, determining length n = len(b_list) = list. extending with spaces a_string = ".join(a_list) g03/demo.goze) g03/demo.goze (list, generating a sequence b_list extend(a_more) g03/demo.goze) g03/demo.goze (list, joining with spaces a_string = ".join(a_list) g03/demo.goze) g03/demo.goze (list, joining with spaces a_string = ".join(a_list) g03/demo.goze) g03/demo.goze (list, gioning with spaces a_string = ".join(a_list) g03/demo.goze) g03/demo.goze (list, gioning with spaces a_string = ".join(a_list) g03/demo.goze) g03/demo.goze (list, gioning with spaces a_string = ".join(a_list) g03/demo.goze) g03/demo.goze (list, gioning with spaces a_string = ".join(a_list) g03/demo.goze) g03/demo.goze (list, gioning with spaces a_string = ".join(a_list) g03/demo.goze) g03/demo.goze (list, gioning with spaces a_string = ".join(a_list) g03/demo.goze) g03/demo.goze (list, gioning with spaces a_string = ".join(a_list) g03/demo.goze) g03/demo.goze (list, gioning with spaces a_string = ".join(a_list) g03/demo.goze) g03/demo.goze (list, gioning with spaces a_string = ".join(a_list) g03/demo.goze) g03/demo.goze (list, gioning with spaces a_string = ".join(a_list) g03/demo.goze (list, gioning with spaces a_string = ".join(a_list	core	file, closing	fh.close()	g02/demo.py
core file, output using print print("It was written during", year, file=fh) core file, output using write fh.write("Where was this file was written?\n") core file, reading one line at a time for line in fh: core for, looping through a list for n in a_list: core function, calling core function, calling core function, defining core function, defining with an optional argument core function, defining with optional argument core function, returning a result core function, returning a result core list, appending an element core list, create via comprehension core list, creating core list, determining length core list, generating a sequence core list, generating a sequence core list, joining with spaces print("It was written during", year, file=fh) fh.write("Where was this file was written?\n") g02/demo.t g04/demo.t g05/demo.t g06/demo.t g06/demo.t g07/demo.t g07/demo.t g07/demo.t g07/demo.t g07/demo.t g07/demo.t g08/demo.t g09/demo.t g09/demo.t g09/demo.t g09/demo.t g09/demo.t g03/demo.t g03/demo.	core	file, opening for reading	fh = open('states.csv')	g05/demo.py
core file, output using write file, write("Where was this file was written?\n") core file, reading one line at a time for line in fh: core for, looping through a list for n in a_list: core function, calling core function, calling with an optional argument core function, defining with optional argument core function, defining with optional argument core function, returning a result core function, returning a result core list, appending an element a_list.append("four") core list, create via comprehension core list, creating a_list = ["zero", "one", "two", "three"] core list, determining length n = len(b_list) core list, generating a sequence b_list = range(1,6) a_string = "".join(a_list) g03/demo.g03/	core	file, opening for writing	fh = open(filename, "w")	g02/demo.py
core file, reading one line at a time for line in fh: core for, looping through a list for n in a_list: core function, calling d1_ssq = sumsq(d1)	core	file, output using print	<pre>print("It was written during",year,file=fh)</pre>	g02/demo.py
core for, looping through a list for n in a_list: core function, calling d1_ssq = sumsq(d1) core function, calling with an optional argument core function, defining def sumsq(values): core function, defining with optional argument core function, defining with optional argument core function, returning a result return values core if statement, inequality test if fips != '00': core list, appending an element a_list.append("four") core list, create via comprehension cubes = [n**3 for n in a_list] core list, creating a_list = ["zero", "one", "two", "three"] core list, determining length n = len(b_list) core list, extending with another list a_list.extend(a_more) list, generating a sequence b_list = range(1,6) core list, joining with spaces d1_ssq = sumsq(d1) g00/demo. g07/demo. g07/demo. g07/demo. g00/demo. g09/demo. g00/demo. g03/demo.	core	file, output using write	fh.write("Where was this file was written?\n")	g02/demo.py
core function, calling d1_ssq = sumsq(d1) g06/demo.recore function, calling with an optional argument core function, defining def sumsq(values): g06/demo.recore function, defining with optional argument core function, defining with optional argument def sample_function(payment,year,r=0.05): g07/demo.recore function, returning a result return values g06/demo.recore if statement, inequality test if fips != '00': g09/demo.recore list, appending an element a_list.append("four") g03/demo.recore list, create via comprehension cubes = [n**3 for n in a_list] g04/demo.recore list, creating a_list = ["zero", "one", "two", "three"] g03/demo.recore list, determining length n = len(b_list) g03/demo.recore list, extending with another list a_list.extend(a_more) g03/demo.recore list, generating a sequence b_list = range(1,6) g04/demo.recore list, joining with spaces a_string = ".join(a_list) g03/demo.recore g03/demo.recore list, joining with spaces a_string = ".join(a_list) g03/demo.recore g03/demo.recore g03/demo.recore list, joining with spaces a_string = ".join(a_list) g03/demo.recore g03/demo.recore g03/demo.recore g03/demo.recore list, joining with spaces a_string = ".join(a_list) g03/demo.recore g03/demo	core	file, reading one line at a time	for line in fh:	g05/demo.py
core function, calling with an optional argument core function, defining def sumsq(values): core function, defining with optional argument core function, returning a result core if statement, inequality test description if fips != '00': core list, appending an element a_list.append("four") core list, create via comprehension cubes = [n**3 for n in a_list] core list, determining length n = len(b_list) core list, extending with another list a_list.extend(a_more) core list, generating a sequence b_list = range(1,6) core list, joining with spaces sample_function(100, 10, r=0.07) g07/demo.p g06/demo.p g07/demo.p g07/demo.p g07/demo.p g07/demo.p g07/demo.p g08/demo.p g09/demo.p g03/demo.p	core	for, looping through a list	for n in a_list:	g04/demo.py
core function, defining def sumsq(values): core function, defining with optional argument def sample_function(payment,year,r=0.05): core function, returning a result return values core if statement, inequality test if fips != '00': core list, appending an element a_list.append("four") core list, create via comprehension cubes = [n**3 for n in a_list] core list, creating a_list = ["zero","one","two","three"] core list, determining length n = len(b_list) core list, extending with another list a_list.extend(a_more) core list, generating a sequence b_list = range(1,6) core list, joining with spaces def sumsq(values): g06/demo.finction(payment,year,r=0.05): g07/demo.finction(payment,year,r=0.05): g08/demo.finction(payment,year,r=0.05): g09/demo.finction(payment,year,r=0.05): g09/demo.finction(payment,year,r=0.05): g07/demo.finction(payment,year,r=0.05): g09/demo.finction(payment,year,r=0.05): g09/demo.finction(payment,year,r=0.05): g00/demo.finction(payment,year,r=0.05): g00/demo.finctio	core	function, calling	$d1_ssq = sumsq(d1)$	g06/demo.py
core function, defining with optional argument core function, returning a result return values g06/demo. g09/demo. g	core	function, calling with an optional argument	sample_function(100, 10, r=0.07)	g07/demo.py
core function, returning a result return values $g06/demo.p$ core if statement, inequality test if fips != '00': $g09/demo.p$ core list, appending an element $g08/demo.p$ a_list_append("four") $g08/demo.p$ core list, create via comprehension $g08/demo.p$ core list, creating $g08/demo.p$ core list, determining length $g08/demo.p$ core list, extending with another list $g08/demo.p$ core list, generating a sequence $g08/demo.p$ core list, generating a sequence $g08/demo.p$ core list, joining with spaces $g08/demo.p$ $g08/$	core		def sumsq(values):	g06/demo.py
core if statement, inequality test if fips $!=$ '00': g09/demo.pcore list, appending an element a_list.append("four") g03/demo.pcore list, create via comprehension cubes = $[n**3 \text{ for n in a_list}]$ g04/demo.pcore list, creating a_list = $["zero","one","two","three"]$ g03/demo.pcore list, determining length n = $len(b_list)$ g03/demo.pcore list, extending with another list a_list.extend(a_more) g03/demo.pcore list, generating a sequence b_list = $range(1,6)$ g04/demo.pcore list, joining with spaces a_string = ".join(a_list) g03/demo.pcore g03/demo.pcor	core	function, defining with optional argument	def sample_function(payment,year,r=0.05):	g07/demo.py
core list, appending an element a_list.append("four") g03/demo.proce list, create via comprehension cubes = $[n**3 \text{ for n in a_list}]$ g04/demo.proce list, creating a_list = $["zero", "one", "two", "three"]$ g03/demo.proce list, determining length n = $len(b_list)$ g03/demo.proce list, extending with another list a_list.extend(a_more) g03/demo.proce list, generating a sequence b_list = $range(1,6)$ g04/demo.proce list, joining with spaces a_string = $range(1,6)$ g03/demo.proce g	core	function, returning a result	return values	g06/demo.py
$ \begin{array}{llllllllllllllllllllllllllllllllllll$	core	if statement, inequality test	if fips != '00':	g09/demo.py
corelist, creating $a_{\text{list}} = [\text{"zero","one","two","three"}]$ $g03/\text{demo.p}$ corelist, determining length $n = \text{len}(b_{\text{list}})$ $g03/\text{demo.p}$ corelist, extending with another list $a_{\text{list.extend}}(a_{\text{more}})$ $g03/\text{demo.p}$ corelist, generating a sequence $b_{\text{list}} = \text{range}(1,6)$ $g04/\text{demo.p}$ corelist, joining with spaces $a_{\text{string}} = \text{"".join}(a_{\text{list}})$ $g03/\text{demo.p}$	core	list, appending an element	a_list.append("four")	g03/demo.py
$ \begin{array}{llllllllllllllllllllllllllllllllllll$	core	list, create via comprehension		g04/demo.py
$ \begin{array}{cccccccccccccccccccccccccccccccccccc$	core			g03/demo.py
core list, generating a sequence $b_{list} = range(1,6)$ $g04/demo.ptg$ core list, joining with spaces $a_{string} = "".join(a_{list})$ $g03/demo.ptg$	core		$n = len(b_list)$	g03/demo.py
core list, joining with spaces $a_string = "".join(a_list)$ $g03/demo.p$	core	list, extending with another list	a_list.extend(a_more)	g03/demo.py
	core	list, generating a sequence		g04/demo.py
core list, selecting an element $print(a_list[0])$ $g03/demo.pti$	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

Module	Description	Example	Script
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, splitting on whitespace	$b_list = b_string.split()$	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)	${ m g05/demo.py}$
core	string, concatenating	name = $s1+""+s2+""+s3$	g02/demo.py
core	string, convert to lower case	line = line.lower()	g05/nato.py
core	string, converting to an int	values.append(int(line))	g06/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, printing	print("Hello, World!")	g02/hello1.py
core	string, remove spaces	line = line.strip()	g05/nato.py
core	string, splitting on white space	parts = line.split(',')	g05/demo.py
core	string, stripping blank space	<pre>clean = [item.strip() for item in parts]</pre>	g05/demo.py
CSV	opening an output file for 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	writing a DictWriter header line	writer.writeheader()	g09/demo.py
CSV	writing a DictWriter row	writer.writerow(name_rec)	g09/demo.py
json	importing the module	import json	g05/demo.py
json	using to print an object nicely	<pre>print(json.dumps(list1,indent=4))</pre>	g05/demo.py
scipy	calling newton's method	${\sf cr = opt.newton(find_cube_root,xinit,maxiter=20,args=[y])}$	g07/demo.py
scipy	importing the module	import scipy.optimize as opt	g07/demo.py