core dictionary, checking for existing key core dictionary, creating core dictionary, creating core dictionary, deleting an entry del name_by_fips("00"] g05/core dictionary, deleting an entry del name_by_fips("00"] g09/core dictionary, iterating over values for rece in name_by_fips.keys(): g09/core dictionary, looking up a value name = ny['name'] g05/core dictionary, making a list of list1 = [co,ny] g05/core dictionary, obtaining a list of keys names = super_dict.keys() g05/core dictionary, obtaining a list of keys names = super_dict.keys() g05/core dictionary, obtaining a list of keys names = super_dict.keys() g05/core file, closing for reading fine open('states.csv') g05/core file, opening for reading fine open('states.csv') g05/core file, opening for writing fine open(filename, "w') g05/core file, output using print print("It was written during", year, file=fine g05/core file, output using write fine write("Where was this file was written?\n") g02/core file, cotyput using write for line in fh: g05/core function, calling with an optional argument core function, defining with optional argument core list, appending an element a_list.append("four") g06/core list, create via comprehension cubes = [n**3 for n in a_list] g09/core list, dermining length n = len(b_list) g03/core list, determining length n = len(b_list) g03/core list, determining length n = len(b_list) g03/core list, generating a sequence b_list = range(1,6) g03/core list, generating a sequence a_a_string = "join(a_list)" g03/core list, joining with spaces a_string = "join(a_list)" g03/core list. g03/core list. g0	Module	Description	Example	Script
core dictionary, checking for existing key of core dictionary, creating core dictionary, deleting an entry del name_by_fips("00"] g05/c core dictionary, deleting an entry del name_by_fips("00"] g09/c core dictionary, iterating over keys for fips in name_by_fips.keys(): g09/c core dictionary, looking up a value name = ny[name"] g05/c core dictionary, making a list of list [co,ny] g05/c core dictionary, obtaining a list of store in a me_by_fips.keys(): g05/c core dictionary, obtaining a list of list [co,ny] g05/c core dictionary, obtaining a list of keys names = super_dict.keys() g05/c core dictionary, obtaining a list of keys names = super_dict.keys() g05/c core file, closing for reading fine open(states.csv') g05/c core file, opening for reading fine open(filename, "w") g05/c core file, opening for writing fine open(filename, "w") g05/c core file, output using print print("It was written during" year,file=fine g05/c core file, output using write fine write("Where was this file was written?\n") g02/c core file, output using write for line in fh: g05/c core function, calling with an optional argument core function, defining with optional argument core list, appending an element list, create via comprehension cubes = [n**3 for n in a_list] g06/c dore list, create via comprehension cubes = [n**3 for n in a_list] g09/c core list, deremining length n = len(b_list) g03/c core list, extending with another list a_list extend(a_more) list, extending with another list a_list extend(a_more) list, extending with appeas a_string = "_ioin(a_list)" g03/c core list, generating a sequence b_list = range(1,6) g03/c core list, generating a sequence a_string = "_ioin(a_list)" g03/c core list, generating a sequence a_string = "_ioin(a_list)" g03/c core list, generating a sequence a_string = "_ioin(a_list)" g03/c core list, generating a sequence a_string = "_ioin(a_list)" g03/c core list, ge	core	dictionary, adding a new entry	co['po'] = 'CO'	g05/demo.py
core dictionary, creating co = {\text{lanme}':\text{Colorado', 'capital':\text{Denver}} \ g05/c core dictionary, iterating over keys for fips in name_by_fips.keys(): g09/c core dictionary, iterating over values for rec in name_by_fips.keys(): g09/c core dictionary, now in a list of list [co,ny] g05/c core dictionary, making a list of list [co,ny] g05/c core dictionary, obtaining a list of keys name = ny[name] g05/c core dictionary, obtaining a list of keys names = super_dict.keys() g05/c core file, closing for reading for reading for eading for reading for eading for eading file, opening for reading file, opening for writing file, opening for writing file, opening for reading file, opening file, opening for reading file, opening file, opening for reading file, opening file,	core	· · · · · · · · · · · · · · · · · · ·	• • •	g09/demo.py
core dictionary, deleting an entry del name_by_fips["00"] g09/core dictionary, iterating over keys for fips in name_by_fips.keys(): g09/core dictionary, iterating over values for rec in name_by_fips.values(): g09/core dictionary, looking up a value name = ny[name] g05/core dictionary, making a list of list1 = [co,ny] g05/core dictionary, obtaining a list of keys names = super_dict.keys() g05/core dictionary, obtaining a list of keys names = super_dict.keys() g05/core file, closing for reading file, close() g02/core file, opening for reading file, opening for writing file, opening for writing file, opening for writing file, opening for writing file, open file, output using print print("It was written during",year,file=file) g02/core file, output using write file, write("Where was this file was written?\n") g02/core file, output using write file for line in fil: g05/core function, calling dil_ssq = sumsq(d1) sample_function (100, 10, r=0.07) def sumsq(values); def sample_function (100, 10, r=0.05): g07/core function, defining with optional argument core function, teturning a result return values g06/core list, appending an element a_list append("four") core list, create via comprehension cubes = [n**3 for n in a_list] g03/core list, creating neight n = [len(b_list)] sa_sist = ["zero","one","two","three"] g03/core list, determining length n = len(b_list) sa_sist = "sioning with spaces" sa_sist = "sioning a_sist =	core			g05/demo.py
core dictionary, iterating over keys for fips in name_by_fips.keys(): core dictionary, looking up a value name = ny[name] core dictionary, looking up a value name = ny[name] core dictionary, looking up a value name = ny[name] g05/core dictionary, obtaining a list of list1 = [co,ny] core dictionary, obtaining a list of keys names = super_dict.keys() g05/core dictionary, obtaining a list of keys names = super_dict.keys() g05/core file, closing fn.close() core file, opening for reading fn = open(states.csv') core file, opening for reiting fn = open(filename,"w") g05/core file, output using print print("It was written during",year,file=fh) g02/core file, output using write fn.write("Where was this file was written?\n") g02/core file, reading one line at a time for line in fh: core function, calling core function, calling core function, defining def sumsq(values): def sample_function(payment,year,r=0.05): g07/core function, fetining core function, returning a result return values core list, appending an element a_list.append("four") core list, create via comprehension cubes = [n**3 for n in a_list] core list, create via comprehension cubes = [n**3 for n in a_list] core list, create via comprehension cubes = [n**3 for n in a_list] core list, create via comprehension cubes = [n**3 for n in a_list] core list, determining length n = len(b_list) a_list = ("zero", "one", "two", "three"] list, estending with another list a_list extend(a_more) b_list = range(1,6) core list, joining with spaces list = [niction, list) g03/core list, joining with spaces list = [niction, list) g03/core list, joining with spaces list = [niction, list) g03/core list, joining with spaces list = [niction, list) g03/core list, joining with spaces list = [niction, list) g03/core list, joining with spaces list = [niction, list) g03/core list, joining with spaces list = [niction, list) g03/core list, joining with spaces	core	dictionary, deleting an entry	·	g09/demo.py
core dictionary, looking up a value name = ny['name'] g05/c core dictionary, making a list of list1 = [co,ny] g05/c core dictionary, making a list of keys names = super_dict.keys() g05/c g05/c core f-string, using a formatting string print(f"PV of {payment} with T={year} and r={r} is \${pv}") g07/c core file, closing for reading fh = open('states.csv') fh = open(filename,"w") g05/c core file, opening for reading fh = open(filename,"w") g02/c core file, output using print print("It was written during", year, file=fh) g02/c core file, output using write fh.write("Where was this file was written?\n") g02/c core file, output using write for line in fh: g05/c core function, calling with an optional argument core function, defining def sumsq(values): def sample_function(100, 10, r=0.07) def sumsq(values): def sample_function(payment, year, r=0.05): g07/c core fire statement, testing for equality if fips == "36": g09/c core list, appending an element a_list.append("four") cubes = [n**3 for n in a_list] g04/c core list, create via comprehension cubes = [n**4" for n in a_list] g04/c core list, create via comprehension cubes = [n**5 for n in a_list] g04/c core list, create via comprehension cubes = [n**5 for n in a_list] g04/c core list, creating length n = len(b_list) n = len(b_list) g03/c core list, generating a sequence b_list = range(1,6) g03/c core list, generating a sequence b_list = range(1,6) g03/c core list, generating a sequence b_list = range(1,6) g03/c core list, joining with spaces	core	dictionary, iterating over keys	for fips in name_by_fips.keys():	g09/demo.py
core dictionary, making a list of 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/c core file, closing file, opening for reading file, opening for writing file, opening for writing file, output using print print("It was written during", year, file=fh) core file, output using write file, was written during", year, file=fh) core file, output using write file, write("Where was this file was written?\n") core function, calling for reading for n in a_list: core function, calling di_sample_function() core function, defining defining def sample_function() core function, defining with optional argument function, returning a result return values core list, appending an element a_list.append("four") core list, create via comprehension cubes [n**3 for n in a_list] core list, create via comprehension cubes [n**3 for n in a_list] core list, creating a sequence b_list erange(1,6) a_list = ["zero","one","three"] a_list = [roin(a_list)) g03/c core list, generating a sequence b_list = range(1,6) a_string = "join(a_list) g04/c g05/c g05/c g05/c g05/c g06/c g06/c g06/c g06/c g06/c g07/c	core	dictionary, iterating over values	for rec in name_by_fips.values():	g09/demo.py
core dictionary, obtaining a list of keys names = super_dict.keys() g05/c core f-string, using a formatting string print(f"PV of {payment} with T={year} and r={r} is \${pv}") g07/c core file, closing fh.close() core file, opening for reading fh = open('states.csv') file, opening for reading fh = open(states.csv') file, opening for writing fh = open(filename, "w") g02/c core file, output using print print("It was written during", year, file=fh) g02/c core file, reading one line at a time for line in fh: core function, calling core function, calling function, calling with an optional argument core function, defining core function, defining with optional argument core function, returning a result core if statement, testing for equality if fips == "36": g09/c core list, appending an element sample_function(payment, year, r=0.05): g07/c core list, creating a_list = preconding frour") cubes = [n**3 for n in a_list] core list, creating a_list = preconding frour, "three"] g03/c core list, determining length n = len(b_list) a_list = range(1.6) a_string = "".join(a_list) g03/c core list, generating a sequence b_list = range(1.6) a_string = "".join(a_list) g04/c core list, joining with spaces	core	dictionary, looking up a value	name = ny['name']	g05/demo.py
core file, closing for reading from the string from the string print from the string print from the string for reading for reading from the string for reading for reading from the string for reading from the string for file, opening for reading from the string for file, opening for writing from the string from the string for file, output using print print from the string for file, output using print print print from the string for file, output using make the string for file, output using make the string for reading one line at a time for line in the string for line in the string for equality from the string for from the string for from the string for equality from the string from the stri	core	dictionary, making a list of	list1 = [co,ny]	g05/demo.py
core file, closing for reading fh.close() g02/c core file, opening for reading fh = open('states.csv') g05/c core file, opening for writing fh = open(filename, "w") g02/c core file, output using print print("It was written during", year, file=fh) g02/c core file, output using write fh.write("Where was this file was written?\n") g02/c core file, reading one line at a time for line in fh: g05/c core for, looping through a list for n in a_list: g04/c core function, calling d1_ssq = sumsq(d1) sample_function(100, 10, r=0.07) g07/c core function, defining def sumsq(values): g06/c core function, defining with an optional argument core function, defining with optional argument def sample_function(payment, year, r=0.05): g07/c core function, returning a result return values g06/c core if statement, testing for equality if fips == "36": g09/c core list, appending an element a_list.append("four") core list, create via comprehension cubes = [n**5 for n in a_list] g04/c core list, creating a_list = ["zero", "one", "two", "three"] g03/c core list, determining length n = len(b_list) g03/c core list, generating a sequence b_list = range(1,6) g04/c core list, joining with spaces a_string = "".join(a_list) g03/c core list, joining with spaces	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 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 function, calling d1_ssq = sumsq(d1) core function, calling with an optional argument core function, defining defsumsq(alues): core function, defining defsample_function(payment, year, r=0.05): core function, returning a result return values 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 length n = len(b_list) core list, generating a sequence b_list = range(1.6) core list, gioning with spaces g05/core list, joining with spaces	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 fh = open(filename, "w") g02/c core file, output using print print("It was written during", year, file=fh) g02/c core file, output using write fh.write("Where was this file was written?\n") g02/c 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 function, defining def sumsq(values): core function, defining def sumsq(values): core function, returning a result def sample_function(payment, year, r=0.05): core if statement, testing for equality if fips == "36": 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, generating a sequence b_list = range(1,6) core list, joining with spaces file, output using print print print("It was written during", year, file=fh) g02/c flt was written during", year, file=fh) g02/c flt was written during", year, file=fh) g02/c flt was written during", year, file=fh) g02/c g04/c g05/c g06/c g06/c g07/c g03/c	core	file, closing	fh.close()	g02/demo.py
core file, output using print print("It was written during", year, file=fh) g02/c core file, output using write for line in fh: g02/c for file, reading one line at a time for line in fh: g05/c for line	core	file, opening for reading	fh = open('states.csv')	g05/demo.py
core file, output using write file, 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 d1_ssq = sumsq(d1) core function, defining with an optional argument core function, defining with optional argument core function, returning a result return values core if statement, testing for equality if fips == "36": 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) list, generating a sequence b_list = range(1,6) core list, joining with spaces for n in a_list: g04/c d1_ssq = sumsq(d1) sample_function(100, 10, r=0.07) g07/c def sumsq(values): def sample_function(payment,year,r=0.05): g07/c def sample_function(payment,year,r=0.05): g07/c def sample_function(payment,year,r=0.05): g07/c core list, appending an element a_list.append("four") core list, create via comprehension cubes = [n**3 for n in a_list] g03/c a_list = ["zero","one","two","three"] core list, generating a sequence b_list = range(1,6) a_string = "".join(a_list) g03/c core list, joining with spaces	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 core function, calling with an optional argument core function, defining def sumsq(values): core function, defining with optional argument core function, returning a result return values core if statement, testing for equality if fips == "36": 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.engel(1,6) core list, generating a sequence b_list = range(1,6) core list, joining with spaces for n in a_list: g04/c core function, calling d1_ssq = sumsq(d1) sample_function(100, 10, r=0.07) def sumsq(values): def sample_function(payment,year,r=0.05): g07/c core function, teturning argument def sample_function(payment,year,r=0.05): g06/c core function, teturning argument def sample_function(payment,year,r=0.05): g07/c core list, appending an element a_list.append("four") g03/c core list, determining length n = len(b_list) g03/c core list, generating a sequence b_list = range(1,6) a_string = ".join(a_list) g03/c core list, joining with spaces	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: g04/c core function, calling d1_ssq = sumsq(d1) core function, defining sample_function(100, 10, r=0.07) core function, defining def sample_function(payment, year, r=0.05): g07/c core function, returning a result return values g06/c core if statement, testing for equality if fips == "36": g09/c 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) list, generating a sequence b_list = range(1,6) core list, joining with spaces d1_ssq = sumsq(d1) g06/c g07/c g0	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/c 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 def sample_function(payment,year,r=0.05): core function, returning a result return values 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 d1_ssq = sumsq(d1) sample_function(100, 10, r=0.07) g07/c g07/c g08/c g09/c g09/	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, defining with optional argument def sample_function(payment,year,r=0.05): core function, returning a result return values 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) def sumsq(values): g06/c def sumsq(values): g06/c def sumsq(values): g07/c def sumsq(values): g08/c g07/c def sumsq(values): g08/c g08/c g09/c core list, appending a result return values g03/c g04/c g03/c g04/c g06/c	core	for, looping through a list	for n in a_list:	g04/demo.py
corefunction, definingdef sumsq(values): $g06/c$ corefunction, defining with optional argumentdef sample_function(payment,year,r=0.05): $g07/c$ corefunction, returning a resultreturn values $g06/c$ corelist, appending an element a_i [ist.append("four") $g03/c$ corelist, create via comprehension $cubes = [n**3 \text{ for n in a_list}]$ $cubes = [n**3 \text{ for n in a_list}]$ corelist, creating a_i [ist = ["zero", "one", "two", "three"] $g03/c$ corelist, determining length n_i = len(b_list) $g03/c$ corelist, extending with another list a_i [ist.extend(a_more) $g03/c$ corelist, generating a sequence b_i [ist = range(1,6) $g04/c$ corelist, joining with spaces a_i string = " "join(a_list) $g03/c$	core	function, calling	$d1_ssq = sumsq(d1)$	g06/demo.py
corefunction, defining with optional argumentdef sample_function(payment,year,r=0.05): $g07/c$ corefunction, returning a resultreturn values $g06/c$ coreif statement, testing for equalityif fips == "36": $g09/c$ corelist, appending an elementa_list.append("four") $g03/c$ corelist, create via comprehensioncubes = $[n**3$ for n in a_list] $g04/c$ corelist, creatinga_list = ["zero", "one", "two", "three"] $g03/c$ corelist, determining lengthn = len(b_list) $g03/c$ corelist, extending with another lista_list.extend(a_more) $g03/c$ corelist, generating a sequenceb_list = range(1,6) $g04/c$ corelist, joining with spacesa_string = " ".join(a_list) $g03/c$	core	function, calling with an optional argument	sample_function(100, 10, r=0.07)	g07/demo.py
core function, returning a result return values $g06/c$ core if statement, testing for equality if fips == "36": $g09/c$ core list, appending an element a_list.append("four") $g03/c$ core list, create via comprehension cubes = $[n^{**}3$ for n in a_list] $g04/c$ core list, creating a_list = $["zero","one","two","three"]$ $g03/c$ core list, determining length n = $[en(b_list)]$ $g03/c$ core list, extending with another list a_list.extend(a_more) $g03/c$ core list, generating a sequence $g03/c$ core list, joining with spaces a_string = ".join(a_list) $g03/c$ g	core	function, defining	def sumsq(values):	g06/demo.py
core if statement, testing for equality if fips == "36": $g09/c$ core list, appending an element a_list.append("four") $g03/c$ core list, create via comprehension cubes = $[n^{**}3 \text{ for n in a_list}]$ $g04/c$ core list, creating a_list = $["zero","one","two","three"]$ $g03/c$ core list, determining length $g03/c$ core list, extending with another list a_list.extend(a_more) $g03/c$ core list, generating a sequence $g03/c$ core list, joining with spaces $g03/c$ $g0$	core	function, defining with optional argument	<pre>def sample_function(payment,year,r=0.05):</pre>	g07/demo.py
core list, appending an element a_list.append("four") $g03/c$ core list, create via comprehension $cubes = [n**3 \text{ for n in a_list}]$ $g04/c$ core list, creating $a_list = ["zero", "one", "two", "three"]$ $g03/c$ core list, determining length $g03/c$ core list, extending with another list $g03/c$ core list, generating a sequence $g03/c$ core list, joining with spaces $g03/c$ g	core	function, returning a result	return values	g06/demo.py
$ \begin{array}{llllllllllllllllllllllllllllllllllll$	core	if statement, testing for equality	if fips == "36":	g09/demo.py
corelist, creating $a_{list} = ["zero", "one", "two", "three"]$ $g03/c$ corelist, determining length $n = len(b_{list})$ $g03/c$ corelist, extending with another list $a_{list} = range(1,6)$ $g03/c$ corelist, generating a sequence $b_{list} = range(1,6)$ $g04/c$ corelist, joining with spaces $a_{list} = range(1,6)$ $g03/c$	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) $g04/c$ core list, joining with spaces a_{-} string = " ".join(a_list) $g03/c$	core		$n = len(b_list)$	g03/demo.py
core list, joining with spaces $a_string = "".join(a_list)$ $g03/c$	core	list, extending with another list	a_list.extend(a_more)	g03/demo.py
	core	list, generating a sequence		g04/demo.py
1	core	· ·	_ , , , ,	g03/demo.py
core list, selecting an element print(a_list[0]) g03/c	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, 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, 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	<pre>clean = [item.strip() for item in parts]</pre>	g05/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	writing a header with DictWriter	writer.writeheader()	g09/demo.py
CSV	writing a record with DictWriter	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	$cr = opt.newton(find_cube_root,xinit,maxiter=20,args=[y])$	g07/demo.py
scipy	importing the module	import scipy.optimize as opt	g07/demo.py