

Python equivalents to AML functions and directives

AML functions [function]

ABS	GETCHAR	JOINFILE*	RESPONSE*
ACCESS	GETCHOICE	KEYWORD	ROUND
ACOS	GETCOVER	LENGTH	SCRACHNAME
AFTER	GETDATABASE	LISTFILE	SEARCH
ANGRAD	GETDATALAYER	LISTIEM	SHOW
ASIN	GETDEFLAYERS	LISTUNIQUE	SIN
ATAN	GETFILE*	LOCASE	SORT*
ATAN2	GETGRID	LOG	SUBST*
BEFORE	GETIMAGE	LOG10	SUBSTR*
CALC	GETITEM	MAX	TAN
CLOSE	GETLAYERCOLS	MENU	TASK
COLUMNINFO	GETLIBRARY	MIN	TOKEN*
COPY	GETSTACK	MOD	TRANSLATE*
COS	GETSYMBOL	NULL	TRIM*
CVTDISTANCE	GETTIN	OKANGLE	TRUNCATE
DATE*	GETUNIQUE	OKDISTANCE	TYPE
DELETE	IACCLOSE	OPEN*	UNQUOTE
DIGNUM	IACCONNECT	PATHNAME	UPCASE
DIR	IACDISCONNECT	QUERY	USERNAME
ENTRYNAME*	IACOPEN	QUOTE	VALUE
EXISTS	IACREQUEST	QUOTEEXISTS	VARIABLE
EXP	INDEX	RADANG	VERIFY
EXTRACT*	INVANGLE	RANDOM*	WRITE
FILELIST	INVDISTANCE	READ	
FORMAT	ITEMINFO	RENAME	

AML directives &directive

ABBREVIATIONS	ENCODE	MENU	STATION
AMLPATH	FLUSHPOINTS	MENUPATH	STOP
ARGS	FORMAT	MESSAGES	SYSTEM
ATOOL	FULLSCREEN	PAUSE	TB
CALL	GETLASTPOINT	POPUP	TERMINAL
CODEPAGE	GETPOINT	PT	TEST
COMMANDS	GOTO	PUSHPOINT	THREAD
CONV_WATCH_TO_AML	IACRETURN	RETURN	TRANSLATE
DALINES	IF THEN ELSE	ROUTINE	TTY
DATA	LABEL	RUN	TYPE
DATEFORMAT	LISTCHAR	RUNWATCH	USAGE
DELVAR	LISTFILES	SELECT	WATCH
DESCRIBE	LISTGLOBAL	SETCHAR	WORKSPACE
DO*	LISTLOCAL	SETVAR	
ECHO	LISTPROGRAM	SEVERITY	
ENABLE	LISTVAR	SHOW	

* Indicates multiple entries

AML functions

AML Function	Python
ABS <x>	abs(x)
ACCESS <path>	import arcgisscripting gp = arcgisscripting.Create() gp.testschemaLock(path)
ACOS <x>	import math math.acos(x)
AFTER <s> <search_s>	s[s.find(search_s) + 1:]
ANGRAD	<not applicable>
ASIN <x>	import math math.asin(x)
ATAN <x>	import math math.atan(x)
ATAN2 <y> <x>	import math math.atan2(y, x)
BEFORE <s> <search_s>	s[:s.find(search_s)]
CALC <expression>	import arcgisscripting gp = arcgisscripting.Create() gp.calculateValue(expression)
CLOSE <file_unit>	file_unit.close()
COLUMNINFO	<not applicable>
COPY <in> <out>	import arcgisscripting gp = arcgisscripting.Create() gp.copy(in, out)
COS <x>	import math math.cos(x)
CVTDISTANCE	<not applicable>
DATE -DEFAULT	import time time.strftime("%Y-%m-%d", localtime())
DATE -FULL	import time time.strftime("%Y-%m-%d.%H:%M:%S.%a", localtime())
DATE -USA	import time time.strftime("%m/%d/%y", localtime())
DATE -UFULL	import time time.strftime("%m/%d/%y.%H:%M:%S.%a", localtime())
DATE -VFULL	import time time.strftime("%d %b %y %H:%M:%S %A", localtime())
DATE -DAY	import time time.strftime("%d", localtime())
DATE -MONTH	import time time.strftime("%B", localtime())
DATE -YEAR	import time time.strftime("%Y", localtime())
DATE -VIS	import time time.strftime("%d %b %y", localtime())
DATE -TIME	import time

	<code>time.strftime("%H:%M:%S", localtime())</code>
DATE -AMPM	<code>import time time.strftime("%I:%M %p", localtime())</code>
DATE -DOW	<code>import time time.strftime("%A", localtime())</code>
DATE -CAL	<code>import time time.strftime("%B %d, %Y", localtime())</code>
DATE -TAG	<code>import time time.strftime("%y%m%d", localtime())</code>
DATE -FTAG	<code>import time time.strftime("%y%m%d.%H%M%S", localtime())</code>
DATE -DFMT	<code>import time time.strftime("format string using below", localtime())</code> %% same as % %a day of week, using locale's abbreviated weekday names %A day of week, using locale's full weekday names %b,%h month, using locale's abbreviated month names %B month, using locale's full month names %c date and time as %x %X %d day of month (01-31) %H hour (00-23) %I hour (00-12) %j day number of year (001-366) %m month number (01-12) %M minute (00-59) %p locale's equivalent of AM or PM, whichever is appropriate %r time as %I:%M:%S %p %S seconds (00-59) %U week number of year (01-52), Sunday is the first day of the week %w day of week; Sunday is day 0 %W week number of year (01-52), Monday is the first %x date, using locale's date format %X time, using locale's time format %y year within century (00-99) %Y year, including century (for example, 1994) %Z time zone abbreviation
DELETE <path>	<code>import arcgisscripting gp = arcgisscripting.Create() gp.delete(path)</code>
DIGNUM	<not applicable>
DIR <path>	<code>import os os.path.dirname(path)</code>
ENTRYNAME <path> -EXT	<code>import os os.path.basename(path)</code>
ENTRYNAME <path> -NOEXT	<code>import os os.path.splitext(path)[0]</code>
ENTRYNAME <path> -EXTONLY	<code>import os os.path.splitext(path)[1]</code>
EXISTS <path>	<code>import arcgisscripting gp = arcgisscripting.Create()</code>

	<code>gp.exists(path)</code>
EXP <x>	<code>import math math.exp(x)</code>
EXTRACT <pos> <elements>	<code>elements.split()[pos - 1]</code> <i>blank seperated</i>
EXTRACT <pos> <elements>	<code>elements.split(",")[pos - 1]</code> <i>comma seperated</i>
FILELIST <specifier>	<code>glob.glob(specifier)</code>
FORMAT <format> {exp1 exp2}	<code>"First exp " + str(exp1) + ", second exp " + str(exp2)</code>
FORMATDATE	<i><not applicable></i>
GETCHAR <prompt>	<code>raw_input[prompt]</code>
GETCHOICE <choices> <prompt> <-SORT>	<pre> from Tkinter import * def PopupList(title, list): root = Tk() root.title(title) root.protocol("WM_DELETE_WINDOW", root.quit) frame = Frame(root) vScrollbar = Scrollbar(frame, orient=VERTICAL) hScrollbar = Scrollbar(frame, orient=HORIZONTAL) listbox = Listbox(frame, selectmode=SINGLE, xscrollcommand=hScrollbar.set, yscrollcommand=vScrollbar.set) vScrollbar.config(command=listbox.yview) vScrollbar.pack(side=RIGHT, fill=Y) hScrollbar.config(command=listbox.xview) hScrollbar.pack(side=BOTTOM, fill=Y) listbox.pack(side=LEFT, fill=BOTH, expand=1) frame.pack() for a in list: listbox.insert(END, a) listbox.bind("<Double-Button-1>", PopupList_callback) listbox.selection_set(0) root.mainloop() index = listbox.curselection() entry = listbox.get(index) root.destroy() return entry def PopupList_callback(event): event.widget.quit() list = choices.split() list.sort() print PopupList(prompt, list) </pre>

GETCOVER <workspace> <wildcard> <prompt> <-SORT>	<pre> from Tkinter import * def PopupList(title, list): root = Tk() root.title(title) root.protocol("WM_DELETE_WINDOW", root.quit) frame = Frame(root) vScrollbar = Scrollbar(frame, orient=VERTICAL) hScrollbar = Scrollbar(frame, orient=HORIZONTAL) listbox = Listbox(frame, selectmode=SINGLE, xscrollcommand=hScrollbar.set, yscrollcommand=vScrollbar.set) vScrollbar.config(command=listbox.yview) vScrollbar.pack(side=RIGHT, fill=Y) hScrollbar.config(command=listbox.xview) hScrollbar.pack(side=BOTTOM, fill=Y) listbox.pack(side=LEFT, fill=BOTH, expand=1) frame.pack() for a in list: listbox.insert(END, a) listbox.bind("<Double-Button-1>", PopupList_callback) listbox.selection_set(0) root.mainloop() index = listbox.curselection() entry = listbox.get(index) root.destroy() return entry def PopupList_callback(event): event.widget.quit() import arcgisscripting gp = arcgisscripting.create() gp.workspace = workspace list = [] pItems = gp.ListDatasets(wildcard, "cover") pItem = pItems.Next() while pItem: list.append(pItem) pItem = pItems.Next() list.sort() print PopupList(prompt, list) </pre>
GETDATABASE	<not applicable>
GETDATALAYER	<not applicable>
GETDEFLAYERS	<not applicable>

```
GETFILE <wildcard> <-INFO>
<prompt> <-SORT>
```

```
from Tkinter import *
def PopupList(title, list):
    root = Tk()
    root.title(title)
    root.protocol("WM_DELETE_WINDOW", root.quit)
    frame = Frame(root)
    vScrollbar = Scrollbar(frame, orient=VERTICAL)
    hScrollbar = Scrollbar(frame, orient=HORIZONTAL)
    listbox = Listbox(frame, selectmode=SINGLE,
xscrollcommand=hScrollbar.set, yscrollcommand=vScrollbar.set)
    vScrollbar.config(command=listbox.yview)
    vScrollbar.pack(side=RIGHT, fill=Y)
    hScrollbar.config(command=listbox.xview)
    hScrollbar.pack(side=BOTTOM, fill=Y)
    listbox.pack(side=LEFT, fill=BOTH, expand=1)
    frame.pack()
    for a in list:
        listbox.insert(END, a)
    listbox.bind("<Double-Button-1>", PopupList_callback)
    listbox.selection_set(0)
    root.mainloop()
    index = listbox.curselection()
    entry = listbox.get(index)
    root.destroy()
    return entry
def PopupList_callback(event):
    event.widget.quit()

import arcgisscripting
gp = arcgisscripting.create()

list = []
pItems = gp.ListTables(wildcard, "INFO")
pItem = pItems.Next()
while pItem:
    list.append(pItem)
    pItem = pItems.Next()
list.sort()
print PopupList(prompt, list)
```

```

GETFILE <wildcard>
<-WORKSPACE> <prompt>
<-SORT>

from Tkinter import *
def PopupList(title, list):
    root = Tk()
    root.title(title)
    root.protocol("WM_DELETE_WINDOW", root.quit)
    frame = Frame(root)
    vScrollbar = Scrollbar(frame, orient=VERTICAL)
    hScrollbar = Scrollbar(frame, orient=HORIZONTAL)
    listbox = Listbox(frame, selectmode=SINGLE,
xscrollcommand=hScrollbar.set, yscrollcommand=vScrollbar.set)
    vScrollbar.config(command=listbox.yview)
    vScrollbar.pack(side=RIGHT, fill=Y)
    hScrollbar.config(command=listbox.xview)
    hScrollbar.pack(side=BOTTOM, fill=Y)
    listbox.pack(side=LEFT, fill=BOTH, expand=1)
    frame.pack()
    for a in list:
        listbox.insert(END, a)
    listbox.bind("<Double-Button-1>", PopupList_callback)
    listbox.selection_set(0)
    root.mainloop()
    index = listbox.curselection()
    entry = listbox.get(index)
    root.destroy()
    return entry
def PopupList_callback(event):
    event.widget.quit()

import arcgisscripting
gp = arcgisscripting.create()

list = []
pItems = gp.ListWorkspaces(wildcard, "COVERAGE")
pItem = pItems.Next()
while pItem:
    list.append(pItem)
    pItem = pItems.Next()
list.sort()
print PopupList(prompt, list)

```

```
GETFILE <wildcard> <-FILE>  
<prompt> <-SORT>
```

```
from Tkinter import *  
def PopupList(title, list):  
    root = Tk()  
    root.title(title)  
    root.protocol("WM_DELETE_WINDOW", root.quit)  
    frame = Frame(root)  
    vScrollbar = Scrollbar(frame, orient=VERTICAL)  
    hScrollbar = Scrollbar(frame, orient=HORIZONTAL)  
    listbox = Listbox(frame, selectmode=SINGLE,  
xscrollcommand=hScrollbar.set, yscrollcommand=vScrollbar.set)  
    vScrollbar.config(command=listbox.yview)  
    vScrollbar.pack(side=RIGHT, fill=Y)  
    hScrollbar.config(command=listbox.xview)  
    hScrollbar.pack(side=BOTTOM, fill=Y)  
    listbox.pack(side=LEFT, fill=BOTH, expand=1)  
    frame.pack()  
    for a in list:  
        listbox.insert(END, a)  
    listbox.bind("<Double-Button-1>", PopupList_callback)  
    listbox.selection_set(0)  
    root.mainloop()  
    index = listbox.curselection()  
    entry = listbox.get(index)  
    root.destroy()  
    return entry  
def PopupList_callback(event):  
    event.widget.quit()  
  
import arcgisscripting, dircache, os  
gp = arcgisscripting.create()  
  
list = []  
dirlist = dircache.listdir(gp.workspace)  
for a in dirlist:  
    if os.path.isfile(a):  
        list.append(a)  
list.sort()  
print PopupList(prompt, list)
```

```
GETFILE <wildcard>  
<-DIRECTORY> <prompt>  
<-SORT>
```

```
from Tkinter import *  
def PopupList(title, list):  
    root = Tk()  
    root.title(title)  
    root.protocol("WM_DELETE_WINDOW", root.quit)  
    frame = Frame(root)  
    vScrollbar = Scrollbar(frame, orient=VERTICAL)  
    hScrollbar = Scrollbar(frame, orient=HORIZONTAL)  
    listbox = Listbox(frame, selectmode=SINGLE,  
xscrollcommand=hScrollbar.set, yscrollcommand=vScrollbar.set)  
    vScrollbar.config(command=listbox.yview)  
    vScrollbar.pack(side=RIGHT, fill=Y)  
    hScrollbar.config(command=listbox.xview)  
    hScrollbar.pack(side=BOTTOM, fill=Y)  
    listbox.pack(side=LEFT, fill=BOTH, expand=1)  
    frame.pack()  
    for a in list:  
        listbox.insert(END, a)  
    listbox.bind("<Double-Button-1>", PopupList_callback)  
    listbox.selection_set(0)  
    root.mainloop()  
    index = listbox.curselection()  
    entry = listbox.get(index)  
    root.destroy()  
    return entry  
def PopupList_callback(event):  
    event.widget.quit()  
  
import arcgisscripting, dircache, os  
gp = arcgisscripting.create()  
  
list = []  
dirlist = dircache.listdir(gp.workspace)  
for a in dirlist:  
    if os.path.isdir(a):  
        list.append(a)  
list.sort()  
print PopupList(prompt, list)
```

```

GETGRID <workspace> <wildcard>
<prompt> <-SORT>

from Tkinter import *
def PopupList(title, list):
    root = Tk()
    root.title(title)
    root.protocol("WM_DELETE_WINDOW", root.quit)
    frame = Frame(root)
    vScrollbar = Scrollbar(frame, orient=VERTICAL)
    hScrollbar = Scrollbar(frame, orient=HORIZONTAL)
    listbox = Listbox(frame, selectmode=SINGLE,
xscrollcommand=hScrollbar.set, yscrollcommand=vScrollbar.set)
    vScrollbar.config(command=listbox.yview)
    vScrollbar.pack(side=RIGHT, fill=Y)
    hScrollbar.config(command=listbox.xview)
    hScrollbar.pack(side=BOTTOM, fill=Y)
    listbox.pack(side=LEFT, fill=BOTH, expand=1)
    frame.pack()
    for a in list:
        listbox.insert(END, a)
    listbox.bind("<Double-Button-1>", PopupList_callback)
    listbox.selection_set(0)
    root.mainloop()
    index = listbox.curselection()
    entry = listbox.get(index)
    root.destroy()
    return entry
def PopupList_callback(event):
    event.widget.quit()

import arcgisscripting
gp = arcgisscripting.create()

gp.workspace = workspace

list = []
pItems = gp.ListRasters(wildcard, "GRID")
pItem = pItems.Next()
while pItem:
    list.append(pItem)
    pItem = pItems.Next()
list.sort()
print PopupList(prompt, list)

```

```
GETIMAGE <workspace>
<wildcard> <type> <prompt>
<-SORT>
```

Type mapping:

```
-ALL use "" (blank)
-BMP use bmp
-TIFF use tiff
-IMAGINE use img
```

```
from Tkinter import *
def PopupList(title, list):
    root = Tk()
    root.title(title)
    root.protocol("WM_DELETE_WINDOW", root.quit)
    frame = Frame(root)
    vScrollbar = Scrollbar(frame, orient=VERTICAL)
    hScrollbar = Scrollbar(frame, orient=HORIZONTAL)
    listbox = Listbox(frame, selectmode=SINGLE,
xscrollcommand=hScrollbar.set, yscrollcommand=vScrollbar.set)
    vScrollbar.config(command=listbox.yview)
    vScrollbar.pack(side=RIGHT, fill=Y)
    hScrollbar.config(command=listbox.xview)
    hScrollbar.pack(side=BOTTOM, fill=Y)
    listbox.pack(side=LEFT, fill=BOTH, expand=1)
    frame.pack()
    for a in list:
        listbox.insert(END, a)
    listbox.bind("<Double-Button-1>", PopupList_callback)
    listbox.selection_set(0)
    root.mainloop()
    index = listbox.curselection()
    entry = listbox.get(index)
    root.destroy()
    return entry
def PopupList_callback(event):
    event.widget.quit()

import arcgisscripting
gp = arcgisscripting.create()

gp.workspace = workspace

list = []
pItems = gp.ListRasters(wildcard, type)
pItem = pItems.Next()
while pItem:
    list.append(pItem)
    pItem = pItems.Next()
list.sort()
print PopupList(prompt, list)
```

<pre>GETITEM <path> <prompt> <-SORT></pre>	<pre>from Tkinter import * def PopupList(title, list): root = Tk() root.title(title) root.protocol("WM_DELETE_WINDOW", root.quit) frame = Frame(root) vScrollbar = Scrollbar(frame, orient=VERTICAL) hScrollbar = Scrollbar(frame, orient=HORIZONTAL) listBox = Listbox(frame, selectmode=SINGLE, xscrollcommand=hScrollbar.set, yscrollcommand=vScrollbar.set) vScrollbar.config(command=listbox.yview) vScrollbar.pack(side=RIGHT, fill=Y) hScrollbar.config(command=listbox.xview) hScrollbar.pack(side=BOTTOM, fill=Y) listBox.pack(side=LEFT, fill=BOTH, expand=1) frame.pack() for a in list: listBox.insert(END, a) listBox.bind("<Double-Button-1>", PopupList_callback) listBox.selection_set(0) root.mainloop() index = listBox.curselection() entry = listBox.get(index) root.destroy() return entry def PopupList_callback(event): event.widget.quit() import arcgisscripting gp = arcgisscripting.create() list = [] pFields = gp.ListFields(path) pField = pFields.Next() while pField: list.append(pField.Name) pField = pFields.Next() list.sort() print PopupList(prompt, list)</pre>
GETLAYERCOLS	<not applicable>
GETLIBRARY	<not applicable>
GETSTACK	<not applicable>
GETSYMBOL	<not applicable>

```
GETTIN <workspace> <wildcard>
<prompt> <-SORT>
```

```
from Tkinter import *
def PopupList(title, list):
    root = Tk()
    root.title(title)
    root.protocol("WM_DELETE_WINDOW", root.quit)
    frame = Frame(root)
    vScrollbar = Scrollbar(frame, orient=VERTICAL)
    hScrollbar = Scrollbar(frame, orient=HORIZONTAL)
    listbox = Listbox(frame, selectmode=SINGLE,
xscrollcommand=hScrollbar.set, yscrollcommand=vScrollbar.set)
    vScrollbar.config(command=listbox.yview)
    vScrollbar.pack(side=RIGHT, fill=Y)
    hScrollbar.config(command=listbox.xview)
    hScrollbar.pack(side=BOTTOM, fill=Y)
    listbox.pack(side=LEFT, fill=BOTH, expand=1)
    frame.pack()
    for a in list:
        listbox.insert(END, a)
    listbox.bind("<Double-Button-1>", PopupList_callback)
    listbox.selection_set(0)
    root.mainloop()
    index = listbox.curselection()
    entry = listbox.get(index)
    root.destroy()
    return entry
def PopupList_callback(event):
    event.widget.quit()

import arcgisscripting
gp = arcgisscripting.create()

gp.workspace = workspace

list = []
pItems = gp.ListDatasets(wildcard, "tin")
pItem = pItems.Next()
while pItem:
    list.append(pItem)
    pItem = pItems.Next()
list.sort()
print PopupList(prompt, list)
```

```
GETUNIQUE <path> <item>
<prompt>
```

```
from Tkinter import *
def PopupList(title, list):
    root = Tk()
    root.title(title)
    root.protocol("WM_DELETE_WINDOW", root.quit)
    frame = Frame(root)
    vScrollbar = Scrollbar(frame, orient=VERTICAL)
    hScrollbar = Scrollbar(frame, orient=HORIZONTAL)
    listbox = Listbox(frame, selectmode=SINGLE,
xscrollcommand=hScrollbar.set, yscrollcommand=vScrollbar.set)
    vScrollbar.config(command=listbox.yview)
    vScrollbar.pack(side=RIGHT, fill=Y)
    hScrollbar.config(command=listbox.xview)
    hScrollbar.pack(side=BOTTOM, fill=Y)
    listbox.pack(side=LEFT, fill=BOTH, expand=1)
    frame.pack()
    for a in list:
        listbox.insert(END, a)
    listbox.bind("<Double-Button-1>", PopupList_callback)
    listbox.selection_set(0)
    root.mainloop()
    index = listbox.curselection()
    entry = listbox.get(index)
    root.destroy()
    return entry
def PopupList_callback(event):
    event.widget.quit()

import arcgisscripting
gp = arcgisscripting.create()

list = []
rows = gp.searchcursor(path)
row = rows.Next()
while row <> None:
    list.append(row.GetValue(item))
    row = rows.Next()

list.sort()
count = len(list)
pos = 1
prev = list[0]
while pos < count:
    if prev == list[pos]:
        del(list[pos])
        count = count - 1
    else:
        prev = list[pos]
        pos = pos + 1
print PopupList(prompt, list)
```

IACCLOSE	<not applicable>
IACCONNECT	<not applicable>
IACDISCONNECT	<not applicable>
IACOPEN	<not applicable>
IACREQUEST	<not applicable>
INDEX <s> <search>	s.find(search) + 1
INVANGLE <x1 y1 x2 y2>	<pre>import math dx = x2 - x1 dy = y2 - y1 if dx == 0.0 and dy == 0.0: azimuth = 0 else: azimuth = math.atan2(math.fabs(dy), math.fabs(dx)) * 57.29577951 if dx < 0.0: azimuth = 180.0 - azimuth if dy < 0.0: azimuth = 360.0 - azimuth</pre>
INVDISTANCE <x1 y1 x2 y2>	<pre>import math dist = math.sqrt((x2-x1)*(x2-x1) + (y2-y1)*(y2-y1))</pre>
ITEMINFO <specifier> <item>	<pre>import arcgisscripting gp = arcgisscripting.create() fields = gp.ListFields(specifier) field = fields.next() while field: if field.name.lower() == item.lower(): print field.name print field.length break field = fields.next()</pre>
JOINFILE <obj1> <obj2> -file -sub	<pre>import os os.path.join(obj1, obj2)</pre>
JOINFILE <obj1> <obj2> -ext	<pre>".".join([obj1.rstrip("."), obj2.lstrip(".")])</pre>
KEYWORD <search_string> <keywords>	<pre>list = keywords.split() count = 1 string_pos = 0 for x in list: if x.find(search_string) == 0: if string_pos == 0: string_pos = count else: string_pos = -1 count = count + 1</pre>
LENGTH <string>	len(string)
LISTFILE	<not applicable>

LISTITEM <specifier>	import arcgisscripting gp = arcgisscripting.create() fields = gp.ListFields(specifier)
LISTUNIQUE <specifier> <item>	import arcgisscripting gp = arcgisscripting.create() sorted(set(getattr(row, item) row in \ iter(gp.SearchCursor(specifier).Next, None)))
LOCASE <s>	s.lower()
LOG <x>	import math math.log(x)
LOG10 <x>	import math math.log10(x)
MAX <x> <y>	max(x, y)
MENU	<not applicable>
MIN <x> <y>	min(x, y)
MOD <x> <y>	x % y
NULL <string>	if string == None or string.strip() == "": ".TRUE." else: ".FALSE."
OKANGLE	<not applicable>
OKDISTANCE	<not applicable>
OPEN <file> -READ	f = open(file, "r")
OPEN <file> -WRITE	f = open(file, "w")
OPEN <file> -APPEND	f = open(file, "a")
PATHNAME <file>	import arcgisscripting, os gp = arcgisscripting.Create() os.path.join(gp.workspace, file)
QUERY	<not applicable>
QUOTE	<not applicable>
QUOTEEXISTS	<not applicable>
RADANG	<not applicable>
RANDOM	import random random.randint(1, 2**31 - 1)
RANDOM <seed>	import random random.seed(seed) random.randint(1, 2**31 - 1)
RANDOM <begin> <end>	import random random.randint(begin, end)
RANDOM <seed> <begin> <end>	import random random.seed(seed) random.randint(begin, end)
READ <file_unit>	file_unit.readline()
RENAME <in> <out>	import arcgisscripting gp = arcgisscripting.Create() gp.rename(in, out)

RESPONSE <prompt> -NOECHO	import getpass getpass.getpass(prompt)
RESPONSE <prompt>	import getpass getpass.default_getpass(prompt)
ROUND <n>	long(round(n))
SCRACHNAME	import arcgisscripting gp = arcgisscripting.Create() gp.createscratchname()
SEARCH <s> <search_s>	s.find(search_s) + 1
SHOW	<not applicable>
SIN <x>	import math math.sin(x)
SORT <elements> -ASCEND	elements.split().sort() <i>blank seperated</i>
SORT <elements> -DESCEND	elements.split().sort(None, None, True) <i>blank seperated</i>
SORT <elements> -ASCEND	elements.split(",").sort() <i>comma seperated</i>
SORT <elements> -DESCEND	elements.split(",").sort(None, None, True) <i>comma seperated</i>
SQRT <x>	import math math.sqrt(x)
SUBST <s> <search_s> <replace_s>	s.replace(search_s, replace_s)
SUBST <s> <search_s>	s.replace(search_s, "")
SUBSTR <s> <pos>	s[pos - 1:]
SUBSTR <s> <pos> <num_chars>	s[pos - 1:pos + num_chars - 1]
TAN <x>	import math math.tan(x)
TASK	<not applicable>
TOKEN <elements> -COUNT	len(elements.split()) <i>blank seperated</i>
TOKEN <elements> -FIND <e>	elements.split().index(e) + 1 <i>blank seperated</i>
TOKEN <elements> -MOVE <from> <to>	e = elements.split()[from - 1] <i>blank seperated</i> del(elements.split()[from - 1]) elements.split().insert(to - 2, e)
TOKEN <elements> -INSERT <pos> <s>	elements.split().insert(pos - 1, s) <i>blank seperated</i>
TOKEN <elements> -DELETE <pos>	del(elements.split()[pos - 1]) <i>blank seperated</i>
TOKEN <elements> -REPLACE <pos> <s>	elements.split()[pos - 1] = s <i>blank seperated</i>
TOKEN <elements> -SWITCH <pos1> <pos2>	e = elements.split()[pos1 - 1] <i>blank seperated</i> elements.split()[pos1 - 1] = elements.split()[pos2 - 1] elements.split()[pos2 - 1] = e
TOKEN <elements> -COUNT	len(elements.split(",")) <i>comma seperated</i>
TOKEN <elements> -FIND <e>	elements.split(",").index(e) + 1 <i>comma seperated</i>
TOKEN <elements> -MOVE <from> <to>	e = elements.split(",")[from - 1] <i>comma seperated</i> del(elements.split(",")[from - 1]) elements.split(",").insert(to - 2, e)
TOKEN <elements> -INSERT <pos> <s>	elements.split(",").insert(pos - 1, s) <i>comma seperated</i>
TOKEN <elements> -DELETE <pos>	del(elements.split(",")[pos - 1]) <i>comma seperated</i>
TOKEN <elements> -REPLACE <pos> <s>	elements.split(",")[pos - 1] = s <i>comma seperated</i>

TOKEN <elements> -SWITCH <pos1> <pos2>	<code>e = elements.split(",")[pos1 - 1]</code> <i>comma separated</i> <code>elements.split(",")[pos1 - 1] = elements.split[pos2 - 1]</code> <code>elements.split(",")[pos2 - 1] = e</code>
TRANSLATE <string>	<code>string.upper()</code>
TRANSLATE <string> <new old>	<code>string.upper()</code> <code>for i in range(min(len(old), len(new))):</code> <code> string.replace(old[i], new[i])</code>
TRIM <s> -BOTH <trim_char>	<code>s.strip(trim_char)</code>
TRIM <s> -LEFT <trim_char>	<code>s.lstrip(trim_char)</code>
TRIM <s> -RIGHT <trim_char>	<code>s.rstrip(trim_char)</code>
TRIM <s> -BOTH	<code>s.strip()</code>
TRIM <s> -LEFT	<code>s.lstrip()</code>
TRIM <s> -RIGHT	<code>s.rstrip()</code>
TRUNCATE <x>	<code>long(x)</code>
TYPE <object>	<code>type(object)</code>
UNQUOTE	<not applicable>
UPCASE <s>	<code>s.upper()</code>
USERNAME	<code>import getpass</code> <code>getpass.getuser()</code>
VALUE	<not applicable>
VARIABLE	<not applicable>
VERIFY <search_string> <string>	<code>count = 0</code> <code>for i in range(len(search_string)):</code> <code> if string.find(search_string[i]) == -1:</code> <code> count = i + 1</code> <code> break</code>
WRITE <file_unit> <string>	<code>file_unit.write(string)</code>

AML directives

Workstation	Python
ABBREVIATIONS	<not supported>
AMPLPATH	<not applicable>
ARGS <var ... var>	<pre>import arcgisscripting gp = arcgisscripting.Create() gp.getparameterastext(var)</pre>
ATOOL	<not applicable>
CALL	<not applicable>
CODEPAGE ANSI	<pre>import locale locale.setlocale(locale.LC_ALL, "")</pre>
COMMANDS	<pre>import arcgisscripting gp = arcgisscripting.create() tools = gp.ListTools() tool = tools.next() while tool: print tool tool = tools.next()</pre>
CONV_WATCH_TO_AML	<not applicable>
DALINES	<not applicable>
DATA	exec or eval statements
DATEFORMAT	Use various functions in the time module. See DATE -DFMT AML function for more information.
DELVAR	del statement
DESCRIBE <geo_dataset>	<pre>import arcgisscripting gp = arcgisscripting.Create() descObj = gp.Describe(geo_dataset) print descObj.Type</pre>
DO LIST	<pre>x = ["a","b","c"] for y in x: print y</pre>
DO REPEAT	<pre>while expression: work</pre>
DO TO BY	<pre>for x in range(0,20,2): print x</pre>
DO UNTIL <expression>	<pre>while not expression: work</pre>
DO WHILE <expression>	<pre>while expression: work</pre>
DO	for or while statements
ECHO	<not applicable>
ENABLE	<not applicable>
ENCODE	<not applicable>
FLUSHPOINTS	<not applicable>

FORMAT	<not applicable>
FULLSCREEN	<not applicable>
GETLASTPOINT	<not applicable>
GETPOINT	<not applicable>
GOTO	<not applicable>
IACRETURN	<not applicable>
IF THEN ELSE	<pre> if expression: True elif expression: True else: False </pre>
LABEL	<not applicable>
LISTCHAR	<not applicable>
LISTFILES	<not applicable>
LISTGLOBAL	<not applicable>
LISTLOCAL	<not applicable>
LISTPROGRAM	<not applicable>
LISTVAR	<not applicable>
MENU	<not applicable>
MENUPATH	<not applicable>
MESSAGES	<not applicable>
PAUSE <prompt> &seconds <secs>	<pre> import time print prompt time.sleep(secs) </pre>
POPUP <file>	<pre> from Tkinter import * def PopupList(title, list): root = Tk() root.title(title) root.protocol("WM_DELETE_WINDOW", root.quit) frame = Frame(root) vScrollbar = Scrollbar(frame, orient=VERTICAL) hScrollbar = Scrollbar(frame, orient=HORIZONTAL) listbox = Listbox(frame, selectmode=SINGLE, xscrollcommand=hScrollbar.set, yscrollcommand=vScrollbar.set) vScrollbar.config(command=listbox.yview) vScrollbar.pack(side=RIGHT, fill=Y) hScrollbar.config(command=listbox.xview) hScrollbar.pack(side=BOTTOM, fill=Y) listbox.pack(side=LEFT, fill=BOTH, expand=1) frame.pack() for a in list: listbox.insert(END, a) root.mainloop() root.destroy() list = [] f = open(file, 'r') line = f.readline() while len(line) <> 0: </pre>

	<pre>list.append(line) line = f.readline() PopupList(file, list)</pre>
PT	<i><use GETMESSAGES to get the execution time of a tool></i>
PUSHPOINT	<i><use geometry object and cursors to create and modify features></i>
RETURN	<i><not applicable></i>
ROUTINE	<i><not applicable></i>
RUN	<i><not applicable></i>
RUNWATCH	<i><not applicable></i>
SELECT	<i><not applicable></i>
SETCHAR	<i><not applicable></i>
SETVAR	<i><not applicable></i>
SEVERITY	<i><not applicable></i>
SHOW	<i><not applicable></i>
STATION	<i><not applicable></i>
STOP	<i><not applicable></i>
SYSTEM <i><command></i>	<pre>import os os.system(command)</pre>
TB	<i><not applicable></i>
TERMINAL	<i><not applicable></i>
TEST	<i><not applicable></i>
THREAD	<i><not applicable></i>
TRANSLATE	<i><not applicable></i>
TTY	<i><not applicable></i>
TYPE <i><message></i>	<pre>print message</pre>
USAGE <i><command></i>	<pre>import arcgisscripting gp = arcgisscripting.Create() gp.usage(command)</pre>
WATCH	<i><not applicable></i>
WORKSPACE <i><path></i>	<pre>import arcgisscripting gp = arcgisscripting.Create() gp.workspace = path</pre>