PYTHON TOOLS/UTILITIES

The standard library comes with a number of modules that can be used both as modules and as command-line utilities.

The *dis* Module:

The dis module is the Python disassembler. It converts byte codes to a format that is slightly more appropriate for human consumption.

You can run the disassembler from the command line. It compiles the given script and prints the disassembled byte codes to the STDOUT. You can also use dis as a module. The **dis** function takes a class, method, function or code object as its single argument.

Example:

```
#!/usr/bin/python
import dis

def sum():
   vara = 10
   varb = 20

   sum = vara + varb
   print "vara + varb = %d" % sum

# Call dis function for the function.

dis.dis(sum)
```

This would produce the following result:

```
  6           0 LOAD_CONST               1 (10)
              3 STORE_FAST               0 (vara)

  7           6 LOAD_CONST               2 (20)
              9 STORE_FAST               1 (varb)

  9          12 LOAD_FAST                0 (vara)
             15 LOAD_FAST                1 (varb)
             18 BINARY_ADD
             19 STORE_FAST               2 (sum)

 10          22 LOAD_CONST               3 ('vara + varb = %d')
             25 LOAD_FAST                2 (sum)
             28 BINARY_MODULO
             29 PRINT_ITEM
             30 PRINT_NEWLINE
             31 LOAD_CONST               0 (None)
             34 RETURN_VALUE
```

The *pdb* Module

The pdb module is the standard Python debugger. It is based on the bdb debugger framework.

You can run the debugger from the command line (type n [or next] to go to the next line and help to get a list of available commands):

Example:

Before you try to run **pdb.py**, set your path properly to Python lib directory. So let us try with above example sum.py:

```
$pdb.py sum.py
> /test/sum.py(3)<module>()
-> import dis
(Pdb) n
> /test/sum.py(5)<module>()
-> def sum():
(Pdb) n
>/test/sum.py(14)<module>()
-> dis.dis(sum)
(Pdb) n
  6           0 LOAD_CONST               1 (10)
              3 STORE_FAST               0 (vara)

  7           6 LOAD_CONST               2 (20)
              9 STORE_FAST               1 (varb)

  9          12 LOAD_FAST                0 (vara)
             15 LOAD_FAST                1 (varb)
             18 BINARY_ADD
             19 STORE_FAST               2 (sum)

 10          22 LOAD_CONST               3 ('vara + varb = %d')
             25 LOAD_FAST                2 (sum)
             28 BINARY_MODULO
             29 PRINT_ITEM
             30 PRINT_NEWLINE
             31 LOAD_CONST               0 (None)
             34 RETURN_VALUE
--Return--
> /test/sum.py(14)<module>()->None
-v dis.dis(sum)
(Pdb) n
--Return--
> <string>(1)<module>()->None
(Pdb)
```

The *profile* Module:

The profile module is the standard Python profiler. You can run the profiler from the command line:

Example:

Let us try to profile the following program:

```
#!/usr/bin/python

vara = 10
varb = 20

sum = vara + varb
print "vara + varb = %d" % sum
```

Now, try running **cProfile.py** over this file *sum.py* as follows:

```
$cProfile.py sum.py
vara + varb = 30
         4 function calls in 0.000 CPU seconds

   Ordered by: standard name

ncalls  tottime  percall  cumtime  percall filename:lineno
   1    0.000    0.000    0.000    0.000 <string>:1(<module>)
   1    0.000    0.000    0.000    0.000 sum.py:3(<module>)
   1    0.000    0.000    0.000    0.000 {execfile}
   1    0.000    0.000    0.000    0.000 {method ......}
```

The *tabnanny* Module

The tabnanny module checks Python source files for ambiguous indentation. If a file mixes tabs and spaces in a

way that throws off indentation, no matter what tab size you're using, the nanny complains:

Example:

Let us try to profile the following program:

```
#!/usr/bin/python

vara = 10
varb = 20

sum = vara + varb
print "vara + varb = %d" % sum
```

If you would try a correct file with tabnanny.py, then it won't complain as follows:

```
$tabnanny.py -v sum.py
'sum.py': Clean bill of health.
```