The dis module All Versions

Examples

What is Python bytecode?

Python is a hybrid interpreter. When running a program, it first assembles it into *bytecode* which can then be run in the Python interpreter (also called a *Python virtual machine*). The dis module in the standard library can be used to make the Python bytecode human-readable by disassembling classes, methods, functions, and code objects.

```
>>> def hello():
...     print "Hello, World"
...
>>> dis.dis(hello)
  2           0 LOAD_CONST               1 ('Hello, World')
              3 PRINT_ITEM
              4 PRINT_NEWLINE
              5 LOAD_CONST               0 (None)
              8 RETURN_VALUE
```

The Python interpreter is stack-based and uses a first-in last-out system.

Each operation code (opcode) in the Python assembly language (the bytecode) takes a fixed number of items from the stack and returns a fixed number of items to the stack. If there aren't enough items on the stack for an opcode, the Python interpreter will crash, possibly without an error message.

Constants in the dis module

```
EXTENDED_ARG = 145 # All opcodes greater than this have 2 operands
HAVE_ARGUMENT = 90 # All opcodes greater than this have at least 1 operands

cmp_op = ('<', '<=', '==', '!=', '>', '>=', 'in', 'not in', 'is', 'is ...
       # A list of comparator id's. The indecies are used as operands in some opcodes

# All opcodes in these lists have the respective types as there operands
hascompare = [107]
hasconst = [100]
hasfree = [135, 136, 137]
hasjabs = [111, 112, 113, 114, 115, 119]
hasjrel = [93, 110, 120, 121, 122, 143]
haslocal = [124, 125, 126]
hasname = [90, 91, 95, 96, 97, 98, 101, 106, 108, 109, 116]

# A map of opcodes to ids
opmap = {'BINARY_ADD': 23, 'BINARY_AND': 64, 'BINARY_DIVIDE': 21, 'BIN...
# A map of ids to opcodes
opname = ['STOP_CODE', 'POP_TOP', 'ROT_TWO', 'ROT_THREE', 'DUP_TOP', '...
```

Disassembling modules

To disassemble a Python module, first this has to be turned into a .pyc file (Python compiled). To do this, run

```
python -m compileall <file>.py
```

Then in an interpreter, run

```
import dis
import marshal
with open("<file>.pyc", "rb") as code_f:
    code_f.read(8) # Magic number and modification time
    code = marshal.load(code_f) # Returns a code object which can be disassembled
    dis.dis(code) # Output the disassembly
```

This will compile a Python module and output the bytecode instructions with dis . The module is never imported so it is safe to use with untrusted code.

Syntax

Parameters

Remarks