Runtime Environment

sys provides low-level APIs for interacting with the system outside of an application, by accepting command line arguments, accessing user input, and passing messages and status values to the user.

Command Line Arguments

The arguments captured by the interpreter are processed there and not passed to the program being run. Any remaining options and arguments, including the name of the script itself, are saved to `sys.argv` in case the program does need to use them.

```
# sys_argv.py

import sys

print('Arguments:', sys.argv)
```

In the third example, the `-u` option is understood by the interpreter, and is not passed to the program being run.

```
$ python3 sys_argv.py

Arguments: ['sys_argv.py']

$ python3 sys_argv.py -v foo blah

Arguments: ['sys_argv.py', '-v', 'foo', 'blah']

$ python3 -u sys_argv.py

Arguments: ['sys_argv.py']
```

See also

- argparse – Module for parsing command line arguments.

Input and Output Streams

Following the Unix paradigm, Python programs can access three file descriptors by default.

```
# sys_stdio.py

import sys

print('STATUS: Reading from stdin', file=sys.stderr)

data = sys.stdin.read()

print('STATUS: Writing data to stdout', file=sys.stderr)

sys.stdout.write(data)
sys.stdout.flush()

print('STATUS: Done', file=sys.stderr)
```

`stdin` is the standard way to read input, usually from a console but also from other programs via a pipeline. `stdout` is the standard way to write output for a user (to the console) or to be sent to the next program in a pipeline. `stderr` is intended for use with warning or error messages.

```
$ cat sys_stdio.py | python3 -u sys_stdio.py

STATUS: Reading from stdin
STATUS: Writing data to stdout
#!/usr/bin/env python3

#end_pymotw_header
import sys

print('STATUS: Reading from stdin', file=sys.stderr)

data = sys.stdin.read()

print('STATUS: Writing data to stdout', file=sys.stderr)

sys.stdout.write(data)
sys.stdout.flush()

print('STATUS: Done', file=sys.stderr)
STATUS: Done
```

See also

- subprocess and pipes – Both subprocess and pipes have features for pipelining programs together.

Returning Status

To return an exit code from a program, pass an integer value to `sys.exit()`.

```
# sys_exit.py
```

```
import sys

exit_code = int(sys.argv[1])
sys.exit(exit_code)
```

A nonzero value means the program exited with an error.

```
$ python3 sys_exit.py 0 ; echo "Exited $?"

Exited 0

$ python3 sys_exit.py 1 ; echo "Exited $?"

Exited 1
```

Interpreter Settings

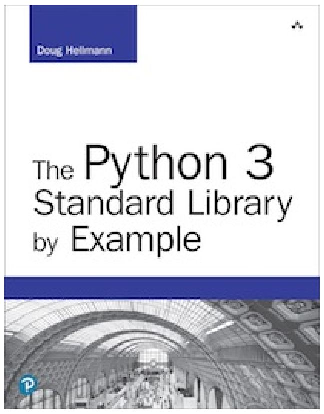
Memory Management and Limits

Quick Links

This page was last updated 2016-12-29.

Navigation

Interpreter Settings
Memory Management and Limits



Get the book

The output from all the example programs from PyMOTW-3 has been generated with Python 3.7.1, unless otherwise noted. Some of the features described here may not be available in earlier versions of Python.

Looking for examples for Python 2?

This Site

Module Index
Index

Other Writing

Blog
The Python Standard Library By Example