Python - Command Line Arguments

Previous Page Next Page

Python provides a **getopt** module that helps you parse command-line options and arguments.

```
$ python test.py arg1 arg2 arg3
```

The Python **sys** module provides access to any command-line arguments via the **sys.argv**. This serves two purposes –

sys.argv is the list of command-line arguments.

len(sys.argv) is the number of command-line arguments.

```
#!/usr/bin/python

import sys

print 'Number of arguments:', len(sys.argv), 'arguments.'
print 'Argument List:', str(sys.argv)
```

Now run above script as follows −

```
$ python test.py arg1 arg2 arg3
```

This produce following result −

```
Number of arguments: 4 arguments.
Argument List: ['test.py', 'arg1', 'arg2', 'arg3']
```

NOTE − As mentioned above, first argument is always script name and it is also being counted in number of arguments.

Parsing Command-Line Arguments

Python provided a **getopt** module that helps you parse command-line options and arguments. This module provides two functions and an exception to enable command line argument parsing.

getopt.getopt method

This method parses command line options and parameter list. Following is simple syntax for this method −

```
getopt.getopt(args, options, [long_options])
```

Here is the detail of the parameters −

This method returns value consisting of two elements: the first is a list of **(option, value)** pairs. The second is the list of program arguments left after the option list was stripped.

Each option-and-value pair returned has the option as its first element, prefixed with a hyphen for short options (e.g., '-x') or two hyphens for long options (e.g., '--long-option').

Exception getopt.GetoptError

This is raised when an unrecognized option is found in the argument list or when an option requiring an argument is given none.

The argument to the exception is a string indicating the cause of the error. The attributes **msg** and **opt** give the error message and related option.

Example

Consider we want to pass two file names through command line and we also want to give an option to check the usage of the script. Usage of the script is as follows −

```
usage: test.py -i <inputfile> -o <outputfile>
```

Here is the following script to test.py −

```
#!/usr/bin/python

import sys, getopt

def main(argv):
   inputfile = ''
   outputfile = ''
   try:
```

```
      elif opt in ("-o", "--ofile"):
         outputfile = arg
   print 'Input file is "', inputfile
   print 'Output file is "', outputfile

if __name__ == "__main__":
   main(sys.argv[1:])
```

Now, run above script as follows –

```
$ test.py -h
usage: test.py -i <inputfile> -o <outputfile>

$ test.py -i BMP -o
usage: test.py -i <inputfile> -o <outputfile>

$ test.py -i inputfile
Input file is " inputfile
Output file is "
```

Previous Page Next Page