Python 3 - Command Line Arguments

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.

Here sys.argv[0] is the program ie. the script name.

Example

Consider the following script **test.py** −

```
#!/usr/bin/python3

import sys

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

Now run the above script as follows −

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

This produces the following result −

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

NOTE − As mentioned above, the first argument is always the 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 the command line options and parameter list. Following is a simple syntax for this method −

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

Here is the detail of the parameters −

- **args** − This is the argument list to be parsed.
- **options** − This is the string of option letters that the script wants to recognize, with options that require an argument should be followed by a colon (:).
- **long_options** − This is an optional parameter and if specified, must be a list of strings with the names of the long options, which should be supported. Long options, which require an argument should be followed by an equal sign ('='). To accept only long options, options should be an empty string.
- This method returns a value consisting of two elements − the first is a list of **(option, value)** pairs, the second is a 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

Suppose 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/python3

import sys, getopt

def main(argv):
   inputfile = ''
   outputfile = ''
   try:
      opts, args = getopt.getopt(argv,"hi:o:",["ifile=","ofile="])
   except getopt.GetoptError:
      print ('test.py -i <inputfile> -o <outputfile>')
      sys.exit(2)
   for opt, arg in opts:
      if opt == '-h':
         print ('test.py -i <inputfile> -o <outputfile>')
         sys.exit()
      elif opt in ("-i", "--ifile"):
         inputfile = arg
      elif opt in ("-o", "--ofile"):
         outputfile = arg
   print ('Input file is "', inputfile)
   print ('Output file is "', outputfile)

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

Output

Now, run the 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 -o outputfile
Input file is " inputfile
Output file is " outputfile
```