

getpass — Secure Password Prompt

Purpose: Prompt the user for a value, usually a password, without echoing what they type to the console.

Many programs that interact with the user via the terminal need to ask the user for password values without showing what the user types on the screen. The `getpass` module provides a portable way to handle such password prompts securely.

Example

The `getpass()` function prints a prompt, then reads input from the user until they press return. The input is returned as a string to the caller.

```
# getpass_defaults.py

import getpass

try:
    p = getpass.getpass()
except Exception as err:
    print('ERROR:', err)
else:
    print('You entered:', p)
```

The default prompt, if none is specified by the caller, is “Password:”.

```
$ python3 getpass_defaults.py

Password:
You entered: sekret
```

The prompt can be changed to any value needed.

```
# getpass_prompt.py

import getpass

p = getpass.getpass(prompt='What is your favorite color? ')
if p.lower() == 'blue':
    print('Right. Off you go.')
else:
    print('Auuuuugh!')
```

Some programs ask for a pass phrase instead of a simple password, to give better security.

```
$ python3 getpass_prompt.py

What is your favorite color?
Right. Off you go.

$ python3 getpass_prompt.py

What is your favorite color?
Auuuuugh!
```

By default, `getpass()` uses `sys.stdout` to print the prompt string. For a program that may produce useful output on `sys.stdout`, it is frequently better to send the prompt to another stream such as `sys.stderr`.

```
# getpass_stream.py

import getpass
import sys
```

```
p = getpass.getpass(stream=sys.stderr)
print('You entered:', p)
```

Using `sys.stderr` for the prompt means standard output can be redirected (to a pipe or file) without seeing the password prompt. The value entered by the user is still not echoed back to the screen.

```
$ python3 getpass_stream.py >/dev/null

Password:
```

Using getpass Without a Terminal

Under Unix, `getpass()` always requires a `tty` it can control via `termios`, so input echoing can be disabled. This means values will not be read from a non-terminal stream redirected to standard input. Instead, `getpass` tries to get to the `tty` for a process, and no error is raised if they can access it.

```
$ echo "not sekret" | python3 getpass_defaults.py

Password:
You entered: sekret
```

It is up to the caller to detect when the input stream is not a `tty`, and use an alternate method for reading in that case.

```
# getpass_noterminal.py

import getpass
import sys

if sys.stdin.isatty():
    p = getpass.getpass('Using getpass: ')
else:
    print('Using readline')
    p = sys.stdin.readline().rstrip()

print('Read: ', p)
```

With a `tty`:

```
$ python3 ./getpass_noterminal.py

Using getpass:
Read:  sekret
```

Without a `tty`:

```
$ echo "sekret" | python3 ./getpass_noterminal.py

Using readline
Read:  sekret
```

See also

- [Standard library documentation for getpass](#)
- [readline](#) – Interactive prompt library.

Quick Links

- [Example](#)
- [Using getpass Without a Terminal](#)

This page was last updated 2016-12-30.

Navigation

- [readline](#) — The GNU readline Library
- [cmd](#) — Line-oriented Command Processors





[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](#)



© Copyright 2019, Doug Hellmann



Other Writing

-  [Blog](#)
-  [The Python Standard Library By Example](#)