Enlighten Documentation

Release 1.1.0

Avram Lubkin

Mar 30, 2019

Contents

CHAPTER 1

PIP

```
$ pip install enlighten
```

CHAPTER 2

RPM

RPMs are available in the Fedora and EPEL repositories

2.1 EL6 and EL7 (RHEL/CentOS/Scientific)

(EPEL repositories must be configured)

```
$ yum install python-enlighten
```

2.2 Fedora

```
$ dnf install python2-enlighten
$ dnf install python3-enlighten
```

CHAPTER 3

Examples

3.1 Basic

For a basic status bar, invoke the *Counter* class directly.

```
import time
import enlighten

pbar = enlighten.Counter(total=100, desc='Basic', unit='ticks')
for num in range(100):
    time.sleep(0.1)  # Simulate work
    pbar.update()
```

3.2 Advanced

To maintain multiple progress bars simultaneously or write to the console, a manager is required.

Advanced output will only work when the output stream, `sys.stdout` by default, is attached to a TTY. *get_manager()* can be used to get a manager instance. It will return a disabled *Manager* instance if the stream is not attached to a TTY and an enabled instance if it is.

```
import time
import enlighten

manager = enlighten.get_manager()
ticks = manager.counter(total=100, desc='Ticks', unit='ticks')
tocks = manager.counter(total=20, desc='Tocks', unit='tocks')

for num in range(100):
    time.sleep(0.1)  # Simulate work
    print(num)
    ticks.update()
```

(continues on next page)

(continued from previous page)

```
    if not num % 5:
        tocks.update()

manager.stop()
```

3.3 Counters

The *Counter* class has two output formats, progress bar and counter.

The progress bar format is used when a total is not None and the count is less than the total. If neither of these conditions are met, the counter format is used:

```
import time
import enlighten

counter = enlighten.Counter(desc='Basic', unit='ticks')
for num in range(100):
    time.sleep(0.1)  # Simulate work
    counter.update()
```

3.4 Additional Examples

- basic - Basic progress bar
- context manager - Managers and counters as context managers
- floats - Support totals and counts that are floats
- multiple with logging - Nested progress bars and logging
- FTP downloader - Show progress downloading files from FTP

3.5 Customization

Enlighten is highly configurable. For information on modifying the output, see the *Series* and *Format* sections of the *Counter* documentation.

CHAPTER 4

Common Patterns

4.1 Enable / Disable

A program may want to disable progress bars based on a configuration setting as well as if output redirection occurs.

```
import sys
import enlighten

# Example configuration object
config = {'stream': sys.stdout,
          'useCounter': False}

enableCounter = config['useCounter'] and stream.isatty()
manager = enlighten.Manager(stream=config['stream'], enabled=enableCounter)
```

The *get_manager()* function slightly simplifies this

```
import enlighten

# Example configuration object
config = {'stream': None,  # Defaults to sys.stdout
          'useCounter': False}

manager = enlighten.get_manager(stream=config['stream'], enabled=config['useCounter'])
```

4.2 Context Managers

Both *Counter* and *Manager* can be used as context managers.

```
import enlighten
```

(continues on next page)

(continued from previous page)

```
SPLINES = 100

with enlighten.Manager() as manager:
    with manager.counter(total=SPLINES, desc='Reticulating:', unit='splines') as␣
↪retic:
        for num in range(SPLINES + 1):
            retic.update()
```

CHAPTER 5

Frequently Asked Questions

5.1 Why is Enlighten called Enlighten?

A progress bar's purpose is to inform the user about an ongoing process. Enlighten, meaning "to inform", seems a fitting name. (Plus any names related to progress were already taken)

5.2 Is Windows supported?

Enlighten is subject to the same limitations as the blessed module which currently doesn't work with windows.

If you have ideas, patches are welcomed.

CHAPTER 6

API Reference

6.1 Classes

class enlighten.**Manager**(*stream=None*, *counter_class=Counter*, ***kwargs*)

Parameters

- **stream** (file object) – Output stream. If `None`, defaults to `sys.stdout`
- **counter_class** (class) – Progress bar class (Default: *`Counter`*)
- **set_scroll** (*`bool`*) – Enable scroll area redefinition (Default: `True`)
- **companion_stream** (file object) – See *companion_stream* below. (Default: `None`)
- **enabled** (*`bool`*) – Status (Default: True)
- **kwargs** (*`dict`*) – Any additional keyword arguments will be used as default values when *`counter()`* is called.

Manager class for outputting progress bars to streams attached to TTYs

Progress bars are displayed at the bottom of the screen with standard output displayed above.

companion_stream

A companion stream is a file object that shares a TTY with the primary output stream. The cursor position in the companion stream will be moved in coordination with the primary stream.

If the value is `None`, `sys.stdout` and `sys.stderr` will be used as companion streams. Unless explicitly specified, a stream which is not attached to a TTY (the case when redirected to a file), will not be used as a companion stream.

counter(*position=None*, ***kwargs*)

Parameters

- **position** (*`int`*) – Line number counting from the bottom of the screen
- **kwargs** (*`dict`*) – Any additional keyword arguments are passed to *`Counter`*

Returns Instance of counter class

Return type *Counter*

Get a new progress bar instance

If `position` is specified, the counter's position can change dynamically if additional counters are called without a `position` argument.

stop()

Clean up and reset terminal

This method should be called when the manager and counters will no longer be needed.

Any progress bars that have `leave` set to `True` or have not been closed will remain on the console. All others will be cleared.

Manager and all counters will be disabled.

class enlighten.**Counter**(***kwargs*)

Parameters

- **bar_format** (*str*) – Progress bar format, see *Format* below
- **count** (*int*) – Initial count (Default: 0)
- **counter_format** (*str*) – Counter format, see *Format* below
- **desc** (*str*) – Description
- **enabled** (*bool*) – Status (Default: True)
- **leave** (*True*) – Leave progress bar after closing (Default: `True`)
- **manager** (*Manager*) – Manager instance. Creates instance if not specified
- **min_delta** (*float*) – Minimum time, in seconds, between refreshes (Default: 0.1)
- **series** (sequence) – Progression series, see *Series* below
- **stream** (file object) – Output stream. Not used when instantiated through a manager
- **total** (*int*) – Total count when complete
- **unit** (*str*) – Unit label

Progress bar and counter class

A *Counter* instance can be created with the *Manager.counter()* method or, when a standalone progress bar for simple applications is required, the *Counter* class can be called directly. The output stream will default to `sys.stdout` unless `stream` is set.

Note: With the default values for `bar_format` and `counter_format`, `floats` can not be used for `total`, `count`, or provided to `update()`. In order to use `floats`, provide custom formats to `bar_format` and `counter_format`. See *Format* below.

Series

The progress bar is constructed from the characters in `series`. `series` must be a sequence (`str`, `list`, `tuple`) containing single characters.

Default progress series (`series`):

```
' '
```

The first character is the fill character. When the `count` is 0, the bar will be made up of only this character. In the example below, characters 5 through 9 are fill characters.

The last character is the full character. When the `count` is equal to `total`, the bar will be made up of only this character. In the example below, characters 0 through 3 are full characters.

The remaining characters are fractional characters used to more accurately represent the transition between the full and fill characters. In the example below, character 4 is a fractional character.

```
'45% |     |'
     '0123456789'
```

Format

If `total` is `None` or `count` becomes higher than `total`, the counter format will be used instead of the progress bar format.

Default counter format (`counter_format`):

```
'{desc}{desc_pad}{count:d} {unit}{unit_pad}{elapsed}, {rate:.2f}{unit_pad}
↪{unit}/s]{fill}'

# Example output
'Loaded 30042 Files [00:01, 21446.45 Files/s]                              ␣
↪       '
```

Default progress bar format (`bar_format`):

```
'{desc}{desc_pad}{percentage:3.0f}%|{bar}| {count:{len_total}d}/{total:d}␣
↪[{elapsed}<{eta}, {rate:.2f}{unit_pad}{unit}/s]'

# Example output
'Processing    22%|                   |  23/101 [00:27<01:32, 0.84 Files/
↪s]'
```

Available fields:

- count(`int`) - Current value of `count`
- desc(`str`) - Value of `desc`
- desc_pad(`str`) - A single space if `desc` is set, otherwise empty
- elapsed(`str`) - Time elapsed since instance was created
- rate(`float`) - Average increments per second since instance was created
- unit(`str`) - Value of `unit`
- unit_pad(`str`) - A single space if `unit` is set, otherwise empty

Addition fields for `bar_format` only:

- bar(`str`) - Progress bar draw with characters from `series`
- eta(`str`) - Estimated time to completion
- len_total(`int`) - Length of `total` when converted to a string
- percentage(`float`) - Percentage complete
- total(`int`) - Value of `total`

Addition fields for `counter_format` only:

- fill(`str`) - blank spaces, number needed to fill line

Instance Attributes

count
`int` - Current count

desc
`str` - Description

elapsed
`float` - Time since start (since last update if `count`equals :py:attr:`total`)

enabled
`bool` - Current status

manager
`Manager` - Manager Instance

position
`int` - Current position

total
`int` - Total count when complete

unit
`str` - Unit label

6.2 Functions

`enlighten.`**`get_manager`**(*stream=None*, *counter_class=Counter*, ***kwargs*)

Parameters

- **`stream`** (file object) – Output stream. If `None`, defaults to `sys.stdout`
- **`counter_class`** (class) – Progress bar class (Default: *`Counter`*)
- **`kwargs`** (*`dict`*) – Any additional keyword arguments will passed to the manager class.

Returns Manager instance

Return type *`Manager`*

Convenience function to get a manager instance

If `stream` is not attached to a TTY, the *`Manager`* instance is disabled.

CHAPTER 7

Overview

Enlighten Progress Bar is a console progress bar module for Python. (Yes, another one.) The main advantage of Enlighten is it allows writing to stdout and stderr without any redirection.

Python Module Index

e

Index