Progress Bar Documentation

Release 3.53.1

Rick van Hattem (Wolph)

Sep 09, 2020

Contents

CHAPTER 1

Examples

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
from __future__ import unicode_literals
from __future__ import with_statement

import functools
import random
import sys
import time

import progressbar

examples = []


def example(fn):
    '''Wrap the examples so they generate readable output'''

    @functools.wraps(fn)
    def wrapped(*args, **kwargs):
        try:
            sys.stdout.write('Running: %s\n' % fn.__name__)
            fn(*args, **kwargs)
            sys.stdout.write('\n')
        except KeyboardInterrupt:
            sys.stdout.write('\nSkipping example.\n\n')
            # Sleep a bit to make killing the script easier
            time.sleep(0.2)

    examples.append(wrapped)
```

(continues on next page)

(continued from previous page)

```
    return wrapped


@example
def fast_example():
    ''' Updates bar really quickly to cause flickering '''
    with progressbar.ProgressBar(widgets=[progressbar.Bar()]) as bar:
        for i in range(100):
            bar.update(int(i / 10), force=True)


@example
def shortcut_example():
    for i in progressbar.progressbar(range(10)):
        time.sleep(0.1)


@example
def prefixed_shortcut_example():
    for i in progressbar.progressbar(range(10), prefix='Hi: '):
        time.sleep(0.1)


@example
def templated_shortcut_example():
    for i in progressbar.progressbar(range(10), suffix='{seconds_elapsed:.1}'):
        time.sleep(0.1)


@example
def with_example_stdout_redirection():
    with progressbar.ProgressBar(max_value=10, redirect_stdout=True) as p:
        for i in range(10):
            if i % 3 == 0:
                print('Some print statement %i' % i)
            # do something
            p.update(i)
            time.sleep(0.1)


@example
def basic_widget_example():
    widgets = [progressbar.Percentage(), progressbar.Bar()]
    bar = progressbar.ProgressBar(widgets=widgets, max_value=10).start()
    for i in range(10):
        # do something
        time.sleep(0.1)
        bar.update(i + 1)
    bar.finish()


@example
def color_bar_example():
    widgets = [
        '\x1b[33mColorful example\x1b[39m',
        progressbar.Percentage(),
        progressbar.Bar(marker='\x1b[32m#\x1b[39m'),
```

(continues on next page)

(continued from previous page)

```
    ]
    bar = progressbar.ProgressBar(widgets=widgets, max_value=10).start()
    for i in range(10):
        # do something
        time.sleep(0.1)
        bar.update(i + 1)
    bar.finish()


@example
def color_bar_animated_marker_example():
    widgets = [
        # Colored animated marker with colored fill:
        progressbar.Bar(marker=progressbar.AnimatedMarker(
            fill='x',
            # fill='',
            fill_wrap='\x1b[32m{}\x1b[39m',
            marker_wrap='\x1b[31m{}\x1b[39m',
        )),
    ]
    bar = progressbar.ProgressBar(widgets=widgets, max_value=10).start()
    for i in range(10):
        # do something
        time.sleep(0.1)
        bar.update(i + 1)
    bar.finish()


@example
def multi_range_bar_example():
    markers = [
        '\x1b[32m█\x1b[39m',  # Done
        '\x1b[33m#\x1b[39m',  # Processing
        '\x1b[31m.\x1b[39m',  # Scheduling
        ' '                   # Not started
    ]
    widgets = [progressbar.MultiRangeBar("amounts", markers=markers)]
    amounts = [0] * (len(markers) - 1) + [25]

    with progressbar.ProgressBar(widgets=widgets, max_value=10).start() as bar:
        while True:
            incomplete_items = [
                idx
                for idx, amount in enumerate(amounts)
                for i in range(amount)
                if idx != 0
            ]
            if not incomplete_items:
                break
            which = random.choice(incomplete_items)
            amounts[which] -= 1
            amounts[which - 1] += 1

            bar.update(amounts=amounts, force=True)
            time.sleep(0.02)
```

(continues on next page)

(continued from previous page)

```
@example
def multi_progress_bar_example(left=True):
    jobs = [
        # Each job takes between 1 and 10 steps to complete
        [0, random.randint(1, 10)]
        for i in range(25)  # 25 jobs total
    ]

    widgets = [
        progressbar.Percentage(),
        ' ', progressbar.MultiProgressBar('jobs', fill_left=left),
    ]

    max_value = sum([total for progress, total in jobs])
    with progressbar.ProgressBar(widgets=widgets, max_value=max_value) as bar:
        while True:
            incomplete_jobs = [
                idx
                for idx, (progress, total) in enumerate(jobs)
                if progress < total
            ]
            if not incomplete_jobs:
                break
            which = random.choice(incomplete_jobs)
            jobs[which][0] += 1
            progress = sum([progress for progress, total in jobs])

            bar.update(progress, jobs=jobs, force=True)
            time.sleep(0.02)


@example
def file_transfer_example():
    widgets = [
        'Test: ', progressbar.Percentage(),
        ' ', progressbar.Bar(marker=progressbar.RotatingMarker()),
        ' ', progressbar.ETA(),
        ' ', progressbar.FileTransferSpeed(),
    ]
    bar = progressbar.ProgressBar(widgets=widgets, max_value=1000).start()
    for i in range(100):
        # do something
        bar.update(10 * i + 1)
    bar.finish()


@example
def custom_file_transfer_example():
    class CrazyFileTransferSpeed(progressbar.FileTransferSpeed):
        '''
        It's bigger between 45 and 80 percent
        '''
        def update(self, bar):
            if 45 < bar.percentage() < 80:
                return 'Bigger Now ' + progressbar.FileTransferSpeed.update(
                    self, bar)
            else:
```

(continues on next page)

(continued from previous page)

```
                return progressbar.FileTransferSpeed.update(self, bar)

    widgets = [
        CrazyFileTransferSpeed(),
        ' <<<', progressbar.Bar(), '>>> ',
        progressbar.Percentage(),
        ' ',
        progressbar.ETA(),
    ]
    bar = progressbar.ProgressBar(widgets=widgets, max_value=1000)
    # maybe do something
    bar.start()
    for i in range(200):
        # do something
        bar.update(5 * i + 1)
    bar.finish()


@example
def double_bar_example():
    widgets = [
        progressbar.Bar('>'), ' ',
        progressbar.ETA(), ' ',
        progressbar.ReverseBar('<'),
    ]
    bar = progressbar.ProgressBar(widgets=widgets, max_value=1000).start()
    for i in range(100):
        # do something
        bar.update(10 * i + 1)
        time.sleep(0.01)
    bar.finish()


@example
def basic_file_transfer():
    widgets = [
        'Test: ', progressbar.Percentage(),
        ' ', progressbar.Bar(marker='0', left='[', right=']'),
        ' ', progressbar.ETA(),
        ' ', progressbar.FileTransferSpeed(),
    ]
    bar = progressbar.ProgressBar(widgets=widgets, max_value=500)
    bar.start()
    # Go beyond the max_value
    for i in range(100, 501, 50):
        time.sleep(0.1)
        bar.update(i)
    bar.finish()


@example
def simple_progress():
    bar = progressbar.ProgressBar(
        widgets=[progressbar.SimpleProgress()],
        max_value=17,
    ).start()
    for i in range(17):
```

(continues on next page)

(continued from previous page)

```
        time.sleep(0.1)
        bar.update(i + 1)
    bar.finish()


@example
def basic_progress():
    bar = progressbar.ProgressBar().start()
    for i in range(10):
        time.sleep(0.1)
        bar.update(i + 1)
    bar.finish()


@example
def progress_with_automatic_max():
    # Progressbar can guess max_value automatically.
    bar = progressbar.ProgressBar()
    for i in bar(range(8)):
        time.sleep(0.1)


@example
def progress_with_unavailable_max():
    # Progressbar can't guess max_value.
    bar = progressbar.ProgressBar(max_value=8)
    for i in bar((i for i in range(8))):
        time.sleep(0.1)


@example
def animated_marker():
    bar = progressbar.ProgressBar(
        widgets=['Working: ', progressbar.AnimatedMarker()])
    for i in bar((i for i in range(5))):
        time.sleep(0.1)


@example
def filling_bar_animated_marker():
    bar = progressbar.ProgressBar(widgets=[
        progressbar.Bar(
            marker=progressbar.AnimatedMarker(fill='#'),
        ),
    ])
    for i in bar(range(15)):
        time.sleep(0.1)


@example
def counter_and_timer():
    widgets = ['Processed: ', progressbar.Counter('Counter: %(value)05d'),
               ' lines (', progressbar.Timer(), ')']
    bar = progressbar.ProgressBar(widgets=widgets)
    for i in bar((i for i in range(15))):
        time.sleep(0.1)
```

(continues on next page)

(continued from previous page)

```
@example
def format_label():
    widgets = [progressbar.FormatLabel(
        'Processed: %(value)d lines (in: %(elapsed)s)')]
    bar = progressbar.ProgressBar(widgets=widgets)
    for i in bar((i for i in range(15))):
        time.sleep(0.1)


@example
def animated_balloons():
    widgets = ['Balloon: ', progressbar.AnimatedMarker(markers='.oO@* ')]
    bar = progressbar.ProgressBar(widgets=widgets)
    for i in bar((i for i in range(24))):
        time.sleep(0.1)


@example
def animated_arrows():
    # You may need python 3.x to see this correctly
    try:
        widgets = ['Arrows: ', progressbar.AnimatedMarker(markers='←↑→↓')]
        bar = progressbar.ProgressBar(widgets=widgets)
        for i in bar((i for i in range(24))):
            time.sleep(0.1)
    except UnicodeError:
        sys.stdout.write('Unicode error: skipping example')


@example
def animated_filled_arrows():
    # You may need python 3.x to see this correctly
    try:
        widgets = ['Arrows: ', progressbar.AnimatedMarker(markers='')]
        bar = progressbar.ProgressBar(widgets=widgets)
        for i in bar((i for i in range(24))):
            time.sleep(0.1)
    except UnicodeError:
        sys.stdout.write('Unicode error: skipping example')


@example
def animated_wheels():
    # You may need python 3.x to see this correctly
    try:
        widgets = ['Wheels: ', progressbar.AnimatedMarker(markers='')]
        bar = progressbar.ProgressBar(widgets=widgets)
        for i in bar((i for i in range(24))):
            time.sleep(0.1)
    except UnicodeError:
        sys.stdout.write('Unicode error: skipping example')


@example
def format_label_bouncer():
    widgets = [
```

(continues on next page)

(continued from previous page)

```
        progressbar.FormatLabel('Bouncer: value %(value)d - '),
        progressbar.BouncingBar(),
    ]
    bar = progressbar.ProgressBar(widgets=widgets)
    for i in bar((i for i in range(100))):
        time.sleep(0.01)


@example
def format_label_rotating_bouncer():
    widgets = [progressbar.FormatLabel('Animated Bouncer: value %(value)d - '),
               progressbar.BouncingBar(marker=progressbar.RotatingMarker())]

    bar = progressbar.ProgressBar(widgets=widgets)
    for i in bar((i for i in range(18))):
        time.sleep(0.1)


@example
def with_right_justify():
    with progressbar.ProgressBar(max_value=10, term_width=20,
                                 left_justify=False) as progress:
        assert progress.term_width is not None
        for i in range(10):
            progress.update(i)


@example
def exceeding_maximum():
    with progressbar.ProgressBar(max_value=1) as progress:
        try:
            progress.update(2)
        except ValueError:
            pass


@example
def reaching_maximum():
    progress = progressbar.ProgressBar(max_value=1)
    try:
        progress.update(1)
    except RuntimeError:
        pass


@example
def stdout_redirection():
    with progressbar.ProgressBar(redirect_stdout=True) as progress:
        print('', file=sys.stdout)
        progress.update(0)


@example
def stderr_redirection():
    with progressbar.ProgressBar(redirect_stderr=True) as progress:
        print('', file=sys.stderr)
        progress.update(0)
```

(continues on next page)

(continued from previous page)

```
@example
def negative_maximum():
    try:
        with progressbar.ProgressBar(max_value=-1) as progress:
            progress.start()
    except ValueError:
        pass


@example
def rotating_bouncing_marker():
    widgets = [progressbar.BouncingBar(marker=progressbar.RotatingMarker())]
    with progressbar.ProgressBar(widgets=widgets, max_value=20,
                                 term_width=10) as progress:
        for i in range(20):
            time.sleep(0.1)
            progress.update(i)

    widgets = [progressbar.BouncingBar(marker=progressbar.RotatingMarker(),
                                       fill_left=False)]
    with progressbar.ProgressBar(widgets=widgets, max_value=20,
                                 term_width=10) as progress:
        for i in range(20):
            time.sleep(0.1)
            progress.update(i)


@example
def incrementing_bar():
    bar = progressbar.ProgressBar(widgets=[
        progressbar.Percentage(),
        progressbar.Bar(),
    ], max_value=10).start()
    for i in range(10):
        # do something
        time.sleep(0.1)
        bar += 1
    bar.finish()


@example
def increment_bar_with_output_redirection():
    widgets = [
        'Test: ', progressbar.Percentage(),
        ' ', progressbar.Bar(marker=progressbar.RotatingMarker()),
        ' ', progressbar.ETA(),
        ' ', progressbar.FileTransferSpeed(),
    ]
    bar = progressbar.ProgressBar(widgets=widgets, max_value=100,
                                  redirect_stdout=True).start()
    for i in range(10):
        # do something
        time.sleep(0.01)
        bar += 10
        print('Got', i)
```

(continues on next page)

(continued from previous page)

```
    bar.finish()


@example
def eta_types_demonstration():
    widgets = [
        progressbar.Percentage(),
        ' ETA: ', progressbar.ETA(),
        ' Adaptive ETA: ', progressbar.AdaptiveETA(),
        ' Absolute ETA: ', progressbar.AbsoluteETA(),
        ' Transfer Speed: ', progressbar.FileTransferSpeed(),
        ' Adaptive Transfer Speed: ', progressbar.AdaptiveTransferSpeed(),
        ' ', progressbar.Bar(),
    ]
    bar = progressbar.ProgressBar(widgets=widgets, max_value=500)
    bar.start()
    for i in range(500):
        if i < 100:
            time.sleep(0.02)
        elif i > 400:
            time.sleep(0.1)
        else:
            time.sleep(0.01)
        bar.update(i + 1)
    bar.finish()


@example
def adaptive_eta_without_value_change():
    # Testing progressbar.AdaptiveETA when the value doesn't actually change
    bar = progressbar.ProgressBar(widgets=[
        progressbar.AdaptiveETA(),
        progressbar.AdaptiveTransferSpeed(),
    ], max_value=2, poll_interval=0.0001)
    bar.start()
    for i in range(100):
        bar.update(1)
        time.sleep(0.1)
    bar.finish()


@example
def iterator_with_max_value():
    # Testing using progressbar as an iterator with a max value
    bar = progressbar.ProgressBar()

    for n in bar(iter(range(100)), 100):
        # iter range is a way to get an iterator in both python 2 and 3
        pass


@example
def eta():
    widgets = [
        'Test: ', progressbar.Percentage(),
        ' | ETA: ', progressbar.ETA(),
        ' | AbsoluteETA: ', progressbar.AbsoluteETA(),
```

(continues on next page)

(continued from previous page)

```
        ' | AdaptiveETA: ', progressbar.AdaptiveETA(),
    ]
    bar = progressbar.ProgressBar(widgets=widgets, max_value=50).start()
    for i in range(50):
        time.sleep(0.1)
        bar.update(i + 1)
    bar.finish()


@example
def variables():
    # Use progressbar.Variable to keep track of some parameter(s) during
    # your calculations
    widgets = [
        progressbar.Percentage(),
        progressbar.Bar(),
        progressbar.Variable('loss'),
        ', ',
        progressbar.Variable('username', width=12, precision=12),
    ]
    with progressbar.ProgressBar(max_value=100, widgets=widgets) as bar:
        min_so_far = 1
        for i in range(100):
            time.sleep(0.01)
            val = random.random()
            if val < min_so_far:
                min_so_far = val
            bar.update(i, loss=min_so_far, username='Some user')


@example
def user_variables():
    tasks = {
        'Download': [
            'SDK',
            'IDE',
            'Dependencies',
        ],
        'Build': [
            'Compile',
            'Link',
        ],
        'Test': [
            'Unit tests',
            'Integration tests',
            'Regression tests',
        ],
        'Deploy': [
            'Send to server',
            'Restart server',
        ],
    }
    num_subtasks = sum(len(x) for x in tasks.values())

    with progressbar.ProgressBar(
            prefix='{variables.task} >> {variables.subtask}',
            variables={'task': '--', 'subtask': '--'},
```

(continues on next page)

(continued from previous page)

```
            max_value=10 * num_subtasks) as bar:
        for tasks_name, subtasks in tasks.items():
            for subtask_name in subtasks:
                for i in range(10):
                    bar.update(bar.value + 1, task=tasks_name,
                               subtask=subtask_name)
                    time.sleep(0.1)


@example
def format_custom_text():
    format_custom_text = progressbar.FormatCustomText(
        'Spam: %(spam).1f kg, eggs: %(eggs)d',
        dict(
            spam=0.25,
            eggs=3,
        ),
    )

    bar = progressbar.ProgressBar(widgets=[
        format_custom_text,
        ' :: ',
        progressbar.Percentage(),
    ])
    for i in bar(range(25)):
        format_custom_text.update_mapping(eggs=i * 2)
        time.sleep(0.1)


@example
def simple_api_example():
    bar = progressbar.ProgressBar(widget_kwargs=dict(fill=''))
    for i in bar(range(200)):
        time.sleep(0.02)


@example
def ETA_on_generators():
    def gen():
        for x in range(200):
            yield None

    widgets = [progressbar.AdaptiveETA(), ' ',
               progressbar.ETA(), ' ',
               progressbar.Timer()]

    bar = progressbar.ProgressBar(widgets=widgets)
    for i in bar(gen()):
        time.sleep(0.02)


@example
def percentage_on_generators():
    def gen():
        for x in range(200):
            yield None
```

(continues on next page)

(continued from previous page)

```
    widgets = [progressbar.Counter(), ' ',
               progressbar.Percentage(), ' ',
               progressbar.SimpleProgress(), ' ']

    bar = progressbar.ProgressBar(widgets=widgets)
    for i in bar(gen()):
        time.sleep(0.02)


def test(*tests):
    if tests:
        for example in examples:

            for test in tests:
                if test in example.__name__:
                    example()
                    break

            else:
                print('Skipping', example.__name__)
    else:
        for example in examples:
            example()


if __name__ == '__main__':
    try:
        test(*sys.argv[1:])
    except KeyboardInterrupt:
        sys.stdout('\nQuitting examples.\n')
```

CHAPTER 2

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

2.1 Types of Contributions

2.1.1 Report Bugs

Report bugs at https://github.com/WoLpH/python-progressbar/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

2.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” is open to whoever wants to implement it.

2.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with “feature” is open to whoever wants to implement it.

2.1.4 Write Documentation

Python Progressbar could always use more documentation, whether as part of the official Python Progressbar docs, in docstrings, or even on the web in blog posts, articles, and such.

2.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/WoLpH/python-progressbar/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

2.2 Get Started!

Ready to contribute? Here's how to set up *python-progressbar* for local development.

1. Fork the *python-progressbar* repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone --branch develop git@github.com:your_name_here/python-progressbar.git
   ```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

   ```
   $ mkvirtualenv progressbar
   $ cd progressbar/
   $ pip install -e .
   ```

4. Create a branch for local development with git-flow-avh:

   ```
   $ git-flow feature start name-of-your-bugfix-or-feature
   ```

   Or without git-flow:

   > $ git checkout -b feature/name-of-your-bugfix-or-feature

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

   ```
   $ flake8 progressbar tests
   $ py.test
   $ tox
   ```

   To get flake8 and tox, just pip install them into your virtualenv using the requirements file.

   > $ pip install -r tests/requirements.txt

6. Commit your changes and push your branch to GitHub with git-flow-avh:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git-flow feature publish
```

Or without git-flow:

$ git add . $ git commit -m “Your detailed description of your changes.” $ git push -u origin feature/name-of-your-bugfix-or-feature

7. Submit a pull request through the GitHub website.

2.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.3, and for PyPy. Check https://travis-ci.org/WoLpH/python-progressbar/pull_requests and make sure that the tests pass for all supported Python versions.

2.4 Tips

To run a subset of tests:

```
$ py.test tests/some_test.py
```

CHAPTER 3

Installation

At the command line:

```
$ pip install progressbar2
```

Or if you don’t have pip:

```
$ easy_install progressbar2
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv progressbar2
$ pip install progressbar2
```

CHAPTER 4

progressbar.shortcuts module

`progressbar.shortcuts.`**`progressbar`**(*iterator*, *min_value=0*, *max_value=None*, *widgets=None*, *prefix=None*, *suffix=None*, ***kwargs*)

CHAPTER 5

progressbar.bar module

class progressbar.bar.**DataTransferBar**(*min_value=0*, *max_value=None*, *widgets=None*, *left_justify=True*, *initial_value=0*, *poll_interval=None*, *widget_kwargs=None*, *custom_len=<function len_color>*, *max_error=True*, *prefix=None*, *suffix=None*, *variables=None*, *min_poll_interval=None*, ***kwargs*)

Bases: `progressbar.bar.ProgressBar`

A progress bar with sensible defaults for downloads etc.

This assumes that the values its given are numbers of bytes.

default_widgets()

class progressbar.bar.**DefaultFdMixin**(*fd=<_io.TextIOWrapper name='<stderr>' mode='w' encoding='UTF-8'>*, *is_terminal=None*, *line_breaks=None*, *enable_colors=None*, ***kwargs*)

Bases: `progressbar.bar.ProgressBarMixinBase`

finish(**args*, ***kwargs*)

update(**args*, ***kwargs*)

class progressbar.bar.**NullBar**(*min_value=0*, *max_value=None*, *widgets=None*, *left_justify=True*, *initial_value=0*, *poll_interval=None*, *widget_kwargs=None*, *custom_len=<function len_color>*, *max_error=True*, *prefix=None*, *suffix=None*, *variables=None*, *min_poll_interval=None*, ***kwargs*)

Bases: `progressbar.bar.ProgressBar`

Progress bar that does absolutely nothing. Useful for single verbosity flags

finish(**args*, ***kwargs*)

Puts the ProgressBar bar in the finished state.

Also flushes and disables output buffering if this was the last progressbar running.

Parameters

- **end** (*str*) – The string to end the progressbar with, defaults to a newline
- **dirty** (*bool*) – When True the progressbar kept the current state and won't be set to 100 percent

start (**args*, ***kwargs*)
Starts measuring time, and prints the bar at 0%.

It returns self so you can use it like this:

Parameters

- **max_value** (*int*) – The maximum value of the progressbar
- **reinit** (*bool*) – Initialize the progressbar, this is useful if you wish to reuse the same progressbar but can be disabled if data needs to be passed along to the next run

```
>>> pbar = ProgressBar().start()
>>> for i in range(100):
...     # do something
...     pbar.update(i+1)
...
>>> pbar.finish()
```

update (**args*, ***kwargs*)
Updates the ProgressBar to a new value.

class progressbar.bar.**ProgressBar**(*min_value=0*, *max_value=None*, *widgets=None*, *left_justify=True*, *initial_value=0*, *poll_interval=None*, *widget_kwargs=None*, *custom_len=<function len_color>*, *max_error=True*, *prefix=None*, *suffix=None*, *variables=None*, *min_poll_interval=None*, ***kwargs*)
Bases: `progressbar.bar.StdRedirectMixin`, `progressbar.bar.ResizableMixin`, `progressbar.bar.ProgressBarBase`

The ProgressBar class which updates and prints the bar.

Parameters

- **min_value** (*int*) – The minimum/start value for the progress bar
- **max_value** (*int*) – The maximum/end value for the progress bar. Defaults to *_DEFAULT_MAXVAL*
- **widgets** (*list*) – The widgets to render, defaults to the result of *default_widget()*
- **left_justify** (*bool*) – Justify to the left if *True* or the right if *False*
- **initial_value** (*int*) – The value to start with
- **poll_interval** (*float*) – The update interval in seconds. Note that if your widgets include timers or animations, the actual interval may be smaller (faster updates). Also note that updates never happens faster than *min_poll_interval* which can be used for reduced output in logs
- **min_poll_interval** (*float*) – The minimum update interval in seconds. The bar will _not_ be updated faster than this, despite changes in the progress, unless *force=True*. This is limited to be at least *_MINIMUM_UPDATE_INTERVAL*. If available, it is also bound by the environment variable PROGRESSBAR_MINIMUM_UPDATE_INTERVAL
- **widget_kwargs** (*dict*) – The default keyword arguments for widgets
- **custom_len** (*function*) – Method to override how the line width is calculated. When using non-latin characters the width calculation might be off by default

- **max_error** (*bool*) – When True the progressbar will raise an error if it goes beyond it's set max_value. Otherwise the max_value is simply raised when needed prefix (str): Prefix the progressbar with the given string suffix (str): Prefix the progressbar with the given string
- **variables** (*dict*) – User-defined variables variables that can be used from a label using *format='{variables.my_var}'*. These values can be updated using *bar.update(my_var='newValue')* This can also be used to set initial values for variables' widgets

A common way of using it is like:

```
>>> progress = ProgressBar().start()
>>> for i in range(100):
...     progress.update(i + 1)
...     # do something
...
>>> progress.finish()
```

You can also use a ProgressBar as an iterator:

```
>>> progress = ProgressBar()
>>> some_iterable = range(100)
>>> for i in progress(some_iterable):
...     # do something
...     pass
...
```

Since the progress bar is incredibly customizable you can specify different widgets of any type in any order. You can even write your own widgets! However, since there are already a good number of widgets you should probably play around with them before moving on to create your own widgets.

The term_width parameter represents the current terminal width. If the parameter is set to an integer then the progress bar will use that, otherwise it will attempt to determine the terminal width falling back to 80 columns if the width cannot be determined.

When implementing a widget's update method you are passed a reference to the current progress bar. As a result, you have access to the ProgressBar's methods and attributes. Although there is nothing preventing you from changing the ProgressBar you should treat it as read only.

Useful methods and attributes include (Public API):

- value: current progress (min_value <= value <= max_value)
- max_value: maximum (and final) value
- end_time: not None if the bar has finished (reached 100%)
- start_time: the time when start() method of ProgressBar was called
- **seconds_elapsed: seconds elapsed since start_time and last call to** update

data()

Returns

- *max_value*: The maximum value (can be None with iterators)
- *start_time*: Start time of the widget
- *last_update_time*: Last update time of the widget
- *end_time*: End time of the widget
- *value*: The current value

- *previous_value*: The previous value
- *updates*: The total update count
- *total_seconds_elapsed*: The seconds since the bar started
- *seconds_elapsed*: The seconds since the bar started modulo 60
- *minutes_elapsed*: The minutes since the bar started modulo 60
- *hours_elapsed*: The hours since the bar started modulo 24
- *days_elapsed*: The hours since the bar started
- *time_elapsed*: The raw elapsed *datetime.timedelta* object
- *percentage*: Percentage as a float or *None* if no max_value is available
- *dynamic_messages*: Deprecated, use *variables* instead.
- *variables*: Dictionary of user-defined variables for the `Variable`'s

Return type dict

default_widgets()

dynamic_messages

finish(*end='\n'*, *dirty=False*)

Puts the ProgressBar bar in the finished state.

Also flushes and disables output buffering if this was the last progressbar running.

Parameters

- **end** (`str`) – The string to end the progressbar with, defaults to a newline
- **dirty** (`bool`) – When True the progressbar kept the current state and won't be set to 100 percent

get_last_update_time()

init()

(re)initialize values to original state so the progressbar can be used (again)

last_update_time

next()

percentage

Return current percentage, returns None if no max_value is given

```
>>> progress = ProgressBar()
>>> progress.max_value = 10
>>> progress.min_value = 0
>>> progress.value = 0
>>> progress.percentage
0.0
>>>
>>> progress.value = 1
>>> progress.percentage
10.0
>>> progress.value = 10
>>> progress.percentage
100.0
>>> progress.min_value = -10
```

(continues on next page)

(continued from previous page)

```
>>> progress.percentage
100.0
>>> progress.value = 0
>>> progress.percentage
50.0
>>> progress.value = 5
>>> progress.percentage
75.0
>>> progress.value = -5
>>> progress.percentage
25.0
>>> progress.max_value = None
>>> progress.percentage
```

set_last_update_time(*value*)

start(*max_value=None*, *init=True*)

Starts measuring time, and prints the bar at 0%.

It returns self so you can use it like this:

Parameters

- **max_value** (*int*) – The maximum value of the progressbar
- **reinit** (*bool*) – Initialize the progressbar, this is useful if you wish to reuse the same progressbar but can be disabled if data needs to be passed along to the next run

```
>>> pbar = ProgressBar().start()
>>> for i in range(100):
...     # do something
...     pbar.update(i+1)
...
>>> pbar.finish()
```

update(*value=None*, *force=False*, ***kwargs*)

Updates the ProgressBar to a new value.

class progressbar.bar.**ProgressBarBase**(***kwargs*)

Bases: collections.abc.Iterable, *progressbar.bar.ProgressBarMixinBase*

class progressbar.bar.**ProgressBarMixinBase**(***kwargs*)

Bases: object

finish()

start(***kwargs*)

update(*value=None*)

class progressbar.bar.**ResizableMixin**(*term_width=None*, ***kwargs*)

Bases: *progressbar.bar.ProgressBarMixinBase*

finish()

class progressbar.bar.**StdRedirectMixin**(*redirect_stderr=False*, *redirect_stdout=False*, ***kwargs*)

Bases: *progressbar.bar.DefaultFdMixin*

finish(*end='\n'*)

start(**args*, ***kwargs*)

`update`(*value=None*)

CHAPTER 6

progressbar.base module

class progressbar.base.**FalseMeta**

Bases: `type`

class progressbar.base.**UnknownLength**

Bases: `object`

CHAPTER 7

progressbar.utils module

class progressbar.utils.**AttributeDict**

Bases: dict

A dict that can be accessed with .attribute

```
>>> attrs = AttributeDict(spam=123)
```

# Reading >>> attrs['spam'] 123 >>> attrs.spam 123

# Read after update using attribute >>> attrs.spam = 456 >>> attrs['spam'] 456 >>> attrs.spam 456

# Read after update using dict access >>> attrs['spam'] = 123 >>> attrs['spam'] 123 >>> attrs.spam 123

# Read after update using dict access >>> del attrs.spam >>> attrs['spam'] Traceback (most recent call last): … KeyError: 'spam' >>> attrs.spam Traceback (most recent call last): … AttributeError: No such attribute: spam >>> del attrs.spam Traceback (most recent call last): … AttributeError: No such attribute: spam

class progressbar.utils.**StreamWrapper**

Bases: object

Wrap stdout and stderr globally

excepthook(*exc_type*, *exc_value*, *exc_traceback*)

flush()

needs_clear()

start_capturing(*bar=None*)

stop_capturing(*bar=None*)

unwrap(*stdout=False*, *stderr=False*)

unwrap_excepthook()

unwrap_stderr()

unwrap_stdout()

update_capturing()

wrap(*stdout=False, stderr=False*)

wrap_excepthook()

wrap_stderr()

wrap_stdout()

class progressbar.utils.**WrappingIO**(*target, capturing=False, listeners={}*)

Bases: object

flush()

flush_target()

write(*value*)

progressbar.utils.**deltas_to_seconds**(**deltas, **kwargs*)

Convert timedeltas and seconds as int to seconds as float while coalescing

```
>>> deltas_to_seconds(datetime.timedelta(seconds=1, milliseconds=234))
1.234
>>> deltas_to_seconds(123)
123.0
>>> deltas_to_seconds(1.234)
1.234
>>> deltas_to_seconds(None, 1.234)
1.234
>>> deltas_to_seconds(0, 1.234)
0.0
>>> deltas_to_seconds()
Traceback (most recent call last):
...
ValueError: No valid deltas passed to `deltas_to_seconds`
>>> deltas_to_seconds(None)
Traceback (most recent call last):
...
ValueError: No valid deltas passed to `deltas_to_seconds`
>>> deltas_to_seconds(default=0.0)
0.0
```

progressbar.utils.**env_flag**(*name, default=None*)

Accepts environt variables formatted as y/n, yes/no, 1/0, true/false, on/off, and returns it as a boolean

If the environt variable is not defined, or has an unknown value, returns *default*

progressbar.utils.**is_ansi_terminal**(*fd, is_terminal=None*)

progressbar.utils.**is_terminal**(*fd, is_terminal=None*)

progressbar.utils.**len_color**(*value*)

Return the length of *value* without ANSI escape codes

```
>>> len_color(b'[1234]abc')
3
>>> len_color(u'[1234]abc')
3
>>> len_color('[1234]abc')
3
```

`progressbar.utils.no_color(value)`

Return the *value* without ANSI escape codes

```
>>> no_color(b'[1234]abc') == b'abc'
True
>>> str(no_color(u'[1234]abc'))
'abc'
>>> str(no_color('[1234]abc'))
'abc'
```

CHAPTER 8

progressbar.widgets module

class progressbar.widgets.**AbsoluteETA**(*format_not_started='Estimated finish time: —-/–/– –:–:–', format_finished='Finished at: %(elapsed)s', format='Estimated finish time: %(eta)s', **kwargs*)

Bases: `progressbar.widgets.ETA`

Widget which attempts to estimate the absolute time of arrival.

class progressbar.widgets.**AdaptiveETA**(***kwargs*)

Bases: `progressbar.widgets.ETA`, `progressbar.widgets.SamplesMixin`

WidgetBase which attempts to estimate the time of arrival.

Uses a sampled average of the speed based on the 10 last updates. Very convenient for resuming the progress halfway.

class progressbar.widgets.**AdaptiveTransferSpeed**(***kwargs*)

Bases: `progressbar.widgets.FileTransferSpeed`, `progressbar.widgets.SamplesMixin`

WidgetBase for showing the transfer speed, based on the last X samples

class progressbar.widgets.**AnimatedMarker**(*markers='|/-\', default=None, fill='', marker_wrap=None, fill_wrap=None, **kwargs*)

Bases: `progressbar.widgets.TimeSensitiveWidgetBase`

An animated marker for the progress bar which defaults to appear as if it were rotating.

class progressbar.widgets.**AutoWidthWidgetBase**(*min_width=None, max_width=None, **kwargs*)

Bases: `progressbar.widgets.WidgetBase`

The base class for all variable width widgets.

This widget is much like the hfill command in TeX, it will expand to fill the line. You can use more than one in the same line, and they will all have the same width, and together will fill the line.

class `progressbar.widgets.`**`Bar`**(*marker='#', left='|', right='|', fill=' ', fill_left=True, marker_wrap=None, **kwargs*)

Bases: `progressbar.widgets.AutoWidthWidgetBase`

A progress bar which stretches to fill the line.

class `progressbar.widgets.`**`BouncingBar`**(*marker='#', left='|', right='|', fill=' ', fill_left=True, marker_wrap=None, **kwargs*)

Bases: `progressbar.widgets.Bar`, `progressbar.widgets.TimeSensitiveWidgetBase`

A bar which has a marker which bounces from side to side.

`INTERVAL = datetime.timedelta(microseconds=100000)`

class `progressbar.widgets.`**`Counter`**(*format='%(value)d', **kwargs*)

Bases: `progressbar.widgets.FormatWidgetMixin`, `progressbar.widgets.WidgetBase`

Displays the current count

class `progressbar.widgets.`**`CurrentTime`**(*format='Current Time: %(current_time)s', microseconds=False, **kwargs*)

Bases: `progressbar.widgets.FormatWidgetMixin`, `progressbar.widgets.TimeSensitiveWidgetBase`

Widget which displays the current (date)time with seconds resolution.

`INTERVAL = datetime.timedelta(seconds=1)`

`current_datetime`()

`current_time`()

class `progressbar.widgets.`**`DataSize`**(*variable='value', format='%(scaled)5.1f %(prefix)s%(unit)s', unit='B', prefixes=('', 'Ki', 'Mi', 'Gi', 'Ti', 'Pi', 'Ei', 'Zi', 'Yi'), **kwargs*)

Bases: `progressbar.widgets.FormatWidgetMixin`, `progressbar.widgets.WidgetBase`

Widget for showing an amount of data transferred/processed.

Automatically formats the value (assumed to be a count of bytes) with an appropriate sized unit, based on the IEC binary prefixes (powers of 1024).

class `progressbar.widgets.`**`DynamicMessage`**(*name, format='{name}: {formatted_value}', width=6, precision=3, **kwargs*)

Bases: `progressbar.widgets.Variable`

Kept for backwards compatibility, please use *Variable* instead.

class `progressbar.widgets.`**`ETA`**(*format_not_started='ETA: --:--:--', format_finished='Time: %(elapsed)8s', format='ETA: %(eta)8s', format_zero='ETA: 00:00:00', format_NA='ETA: N/A', **kwargs*)

Bases: `progressbar.widgets.Timer`

WidgetBase which attempts to estimate the time of arrival.

class `progressbar.widgets.`**`FileTransferSpeed`**(*format='%(scaled)5.1f %(prefix)s%(unit)-s/s', inverse_format='%(scaled)5.1f s/%(prefix)s%(unit)-s', unit='B', prefixes=('', 'Ki', 'Mi', 'Gi', 'Ti', 'Pi', 'Ei', 'Zi', 'Yi'), **kwargs*)

Bases: `progressbar.widgets.FormatWidgetMixin`, `progressbar.widgets.TimeSensitiveWidgetBase`

WidgetBase for showing the transfer speed (useful for file transfers).

class progressbar.widgets.**FormatCustomText** (*format*, *mapping={}*, ***kwargs*)

Bases: *progressbar.widgets.FormatWidgetMixin*, *progressbar.widgets.WidgetBase*

copy = False

mapping = {}

update_mapping (***mapping*)

class progressbar.widgets.**FormatLabel** (*format*, ***kwargs*)

Bases: *progressbar.widgets.FormatWidgetMixin*, *progressbar.widgets.WidgetBase*

Displays a formatted label

```
>>> label = FormatLabel('%(value)s', min_width=5, max_width=10)
>>> class Progress(object):
...     pass
>>> label = FormatLabel('{value} :: {value:^6}', new_style=True)
>>> str(label(Progress, dict(value='test')))
'test ::  test '
```

mapping = {'elapsed': ('total_seconds_elapsed', <function format_time>), 'finished':

class progressbar.widgets.**FormatWidgetMixin** (*format*, *new_style=False*, ***kwargs*)

Bases: object

Mixin to format widgets using a formatstring

Variables available:

- max_value: The maximum value (can be None with iterators)
- value: The current value
- total_seconds_elapsed: The seconds since the bar started
- seconds_elapsed: The seconds since the bar started modulo 60
- minutes_elapsed: The minutes since the bar started modulo 60
- hours_elapsed: The hours since the bar started modulo 24
- days_elapsed: The hours since the bar started
- time_elapsed: Shortcut for HH:MM:SS time since the bar started including days
- percentage: Percentage as a float

required_values = []

class progressbar.widgets.**MultiProgressBar** (*name*, *markers=' '*, ***kwargs*)

Bases: *progressbar.widgets.MultiRangeBar*

get_values (*progress*, *data*)

class progressbar.widgets.**MultiRangeBar** (*name*, *markers*, ***kwargs*)

Bases: *progressbar.widgets.Bar*, *progressbar.widgets.VariableMixin*

A bar with multiple sub-ranges, each represented by a different symbol

The various ranges are represented on a user-defined variable, formatted as

```
[
    ['Symbol1', amount1],
    ['Symbol2', amount2],
```

(continues on next page)

(continued from previous page)

```
    ...
]
```

get_values (*progress, data*)

class progressbar.widgets.**Percentage** (*format='%(percentage)3d%%', **kwargs*)

Bases: *progressbar.widgets.FormatWidgetMixin*, *progressbar.widgets.WidgetBase*

Displays the current percentage as a number with a percent sign.

class progressbar.widgets.**ReverseBar** (*marker='#', left='|', right='|', fill=' ', fill_left=False, **kwargs*)

Bases: *progressbar.widgets.Bar*

A bar which has a marker that goes from right to left

progressbar.widgets.**RotatingMarker**

alias of *progressbar.widgets.AnimatedMarker*

class progressbar.widgets.**SamplesMixin** (*samples=datetime.timedelta(seconds=2), key_prefix=None, **kwargs*)

Bases: *progressbar.widgets.TimeSensitiveWidgetBase*

Mixing for widgets that average multiple measurements

Note that samples can be either an integer or a timedelta to indicate a certain amount of time

```
>>> class progress:
...     last_update_time = datetime.datetime.now()
...     value = 1
...     extra = dict()
```

```
>>> samples = SamplesMixin(samples=2)
>>> samples(progress, None, True)
(None, None)
>>> progress.last_update_time += datetime.timedelta(seconds=1)
>>> samples(progress, None, True) == (datetime.timedelta(seconds=1), 0)
True
```

```
>>> progress.last_update_time += datetime.timedelta(seconds=1)
>>> samples(progress, None, True) == (datetime.timedelta(seconds=1), 0)
True
```

```
>>> samples = SamplesMixin(samples=datetime.timedelta(seconds=1))
>>> _, value = samples(progress, None)
>>> value
[1, 1]
```

```
>>> samples(progress, None, True) == (datetime.timedelta(seconds=1), 0)
True
```

get_sample_times (*progress, data*)

get_sample_values (*progress, data*)

class progressbar.widgets.**SimpleProgress** (*format='%(value_s)s of %(max_value_s)s', **kwargs*)

Bases: *progressbar.widgets.FormatWidgetMixin*, *progressbar.widgets.WidgetBase*

Returns progress as a count of the total (e.g.: "5 of 47")

DEFAULT_FORMAT = '%(value_s)s of %(max_value_s)s'

class progressbar.widgets.**TimeSensitiveWidgetBase**(*min_width=None*, *max_width=None*, ***kwargs*)

Bases: *progressbar.widgets.WidgetBase*

The base class for all time sensitive widgets.

Some widgets like timers would become out of date unless updated at least every *INTERVAL*

INTERVAL = datetime.timedelta(microseconds=100000)

class progressbar.widgets.**Timer**(*format='Elapsed Time: %(elapsed)s'*, ***kwargs*)

Bases: *progressbar.widgets.FormatLabel*, *progressbar.widgets.TimeSensitiveWidgetBase*

WidgetBase which displays the elapsed seconds.

static format_time(*timestamp*, *precision=datetime.timedelta(seconds=1)*)

Formats timedelta/datetime/seconds

```
>>> format_time('1')
'0:00:01'
>>> format_time(1.234)
'0:00:01'
>>> format_time(1)
'0:00:01'
>>> format_time(datetime.datetime(2000, 1, 2, 3, 4, 5, 6))
'2000-01-02 03:04:05'
>>> format_time(datetime.date(2000, 1, 2))
'2000-01-02'
>>> format_time(datetime.timedelta(seconds=3661))
'1:01:01'
>>> format_time(None)
'--:--:--'
>>> format_time(format_time)  # doctest: +ELLIPSIS
Traceback (most recent call last):
    ...
TypeError: Unknown type ...
```

class progressbar.widgets.**Variable**(*name*, *format='{name}: {formatted_value}'*, *width=6*, *precision=3*, ***kwargs*)

Bases: *progressbar.widgets.FormatWidgetMixin*, *progressbar.widgets.VariableMixin*, *progressbar.widgets.WidgetBase*

Displays a custom variable.

class progressbar.widgets.**VariableMixin**(*name*, ***kwargs*)

Bases: object

Mixin to display a custom user variable

class progressbar.widgets.**WidgetBase**(*min_width=None*, *max_width=None*, ***kwargs*)

Bases: *progressbar.widgets.WidthWidgetMixin*

copy = True

class progressbar.widgets.**WidthWidgetMixin**(*min_width=None*, *max_width=None*, ***kwargs*)

Bases: object

Mixing to make sure widgets are only visible if the screen is within a specified size range so the progressbar fits on both large and small screens..

Variables available:

- min_width: Only display the widget if at least *min_width* is left
- max_width: Only display the widget if at most *max_width* is left

```
>>> class Progress(object):
...     term_width = 0
```

```
>>> WidthWidgetMixin(5, 10).check_size(Progress)
False
>>> Progress.term_width = 5
>>> WidthWidgetMixin(5, 10).check_size(Progress)
True
>>> Progress.term_width = 10
>>> WidthWidgetMixin(5, 10).check_size(Progress)
True
>>> Progress.term_width = 11
>>> WidthWidgetMixin(5, 10).check_size(Progress)
False
```

check_size(*progress*)

progressbar.widgets.**create_marker**(*marker*, *wrap=None*)

progressbar.widgets.**create_wrapper**(*wrapper*)

Convert a wrapper tuple or format string to a format string

```
>>> create_wrapper('')
```

```
>>> print(create_wrapper('a{}b'))
a{}b
```

```
>>> print(create_wrapper(('a', 'b')))
a{}b
```

progressbar.widgets.**string_or_lambda**(*input_*)

progressbar.widgets.**wrapper**(*function*, *wrapper*)

Wrap the output of a function in a template string or a tuple with begin/end strings

CHAPTER 9

Text progress bar library for Python.

Travis status:

Coverage:

9.1 Install

The package can be installed through *pip* (this is the recommended method):

> pip install progressbar2

Or if *pip* is not available, *easy_install* should work as well:

> easy_install progressbar2

Or download the latest release from Pypi (https://pypi.python.org/pypi/progressbar2) or Github.

Note that the releases on Pypi are signed with my GPG key (https://pgp.mit.edu/pks/lookup?op=vindex&search=0xE81444E9CE1F695D) and can be checked using GPG:

> gpg –verify progressbar2-<version>.tar.gz.asc progressbar2-<version>.tar.gz

9.2 Introduction

A text progress bar is typically used to display the progress of a long running operation, providing a visual cue that processing is underway.

The ProgressBar class manages the current progress, and the format of the line is given by a number of widgets. A widget is an object that may display differently depending on the state of the progress bar. There are many types of widgets:

- AbsoluteETA
- AdaptiveETA
- AdaptiveTransferSpeed
- AnimatedMarker
- Bar
- BouncingBar
- Counter
- CurrentTime
- DataSize
- DynamicMessage
- ETA
- FileTransferSpeed
- FormatCustomText
- FormatLabel
- Percentage
- ReverseBar
- RotatingMarker
- SimpleProgress
- Timer

The progressbar module is very easy to use, yet very powerful. It will also automatically enable features like auto-resizing when the system supports it.

9.3 Known issues

Due to limitations in both the IDLE shell and the Jetbrains (Pycharm) shells this progressbar cannot function properly within those.

- The IDLE editor doesn't support these types of progress bars at all: https://bugs.python.org/issue23220
- The Jetbrains (Pycharm) editors partially work but break with fast output. As a workaround make sure you only write to either *sys.stdout* (regular print) or *sys.stderr* at the same time. If you do plan to use both, make sure you wait about ~200 milliseconds for the next output or it will break regularly. Linked issue: https://github.com/WoLpH/python-progressbar/issues/115
- Jupyter notebooks buffer *sys.stdout* which can cause mixed output. This issue can be resolved easily using: *import sys; sys.stdout.flush()*. Linked issue: https://github.com/WoLpH/python-progressbar/issues/173

9.4 Links

- **Documentation**
  - https://progressbar-2.readthedocs.org/en/latest/
- **Source**

    - https://github.com/WoLpH/python-progressbar

- **Bug reports**
    - https://github.com/WoLpH/python-progressbar/issues
- **Package homepage**
    - https://pypi.python.org/pypi/progressbar2
- **My blog**
    - https://w.wol.ph/

9.5 Usage

There are many ways to use Python Progressbar, you can see a few basic examples here but there are many more in the examples file.

9.5.1 Wrapping an iterable

```
import time
import progressbar

for i in progressbar.progressbar(range(100)):
    time.sleep(0.02)
```

9.5.2 Progressbars with logging

Progressbars with logging require *stderr* redirection _before_ the *StreamHandler* is initialized. To make sure the *stderr* stream has been redirected on time make sure to call *progressbar.streams.wrap_stderr()* before you initialize the *logger*.

One option to force early initialization is by using the *WRAP_STDERR* environment variable, on Linux/Unix systems this can be done through:

```
# WRAP_STDERR=true python your_script.py
```

If you need to flush manually while wrapping, you can do so using:

```
import progressbar

progressbar.streams.flush()
```

In most cases the following will work as well, as long as you initialize the *StreamHandler* after the wrapping has taken place.

```
import time
import logging
import progressbar

progressbar.streams.wrap_stderr()
logging.basicConfig()

```

(continues on next page)

(continued from previous page)

```
for i in progressbar.progressbar(range(10)):
    logging.error('Got %d', i)
    time.sleep(0.2)
```

9.5.3 Context wrapper

```
import time
import progressbar

with progressbar.ProgressBar(max_value=10) as bar:
    for i in range(10):
        time.sleep(0.1)
        bar.update(i)
```

9.5.4 Combining progressbars with print output

```
import time
import progressbar

for i in progressbar.progressbar(range(100), redirect_stdout=True):
    print('Some text', i)
    time.sleep(0.1)
```

9.5.5 Progressbar with unknown length

```
import time
import progressbar

bar = progressbar.ProgressBar(max_value=progressbar.UnknownLength)
for i in range(20):
    time.sleep(0.1)
    bar.update(i)
```

9.5.6 Bar with custom widgets

```
import time
import progressbar

widgets=[
    ' [', progressbar.Timer(), '] ',
    progressbar.Bar(),
    ' (', progressbar.ETA(), ') ',
]
for i in progressbar.progressbar(range(20), widgets=widgets):
    time.sleep(0.1)
```

9.5.7 Bar with wide Chinese (or other multibyte) characters

```
# vim: fileencoding=utf-8
import time
import progressbar


def custom_len(value):
    # These characters take up more space
    characters = {
        '': 2,
        '': 2,
    }

    total = 0
    for c in value:
        total += characters.get(c, 1)

    return total


bar = progressbar.ProgressBar(
    widgets=[
        ': ',
        progressbar.Bar(),
        ' ',
        progressbar.Counter(format='%(value)02d/%(max_value)d'),
    ],
    len_func=custom_len,
)
for i in bar(range(10)):
    time.sleep(0.1)
```

9.6 Indices and tables

- genindex
- modindex
- search

Python Module Index

Index