
Progress Bar Documentation

Release 4.4.2

Rick van Hattem (Wolph)

Mar 04, 2024

CONTENTS

1	Usage	1
1.1	Wrapping an iterable	1
1.2	Context wrapper	1
1.3	Combining progressbars with print output	1
1.4	Progressbar with unknown length	2
1.5	Bar with custom widgets	2
2	Examples	3
3	Contributing	21
3.1	Types of Contributions	21
3.1.1	Report Bugs	21
3.1.2	Fix Bugs	21
3.1.3	Implement Features	21
3.1.4	Write Documentation	21
3.1.5	Submit Feedback	22
3.2	Get Started!	22
3.3	Pull Request Guidelines	23
3.4	Tips	23
4	Installation	25
5	progressbar.shortcuts module	27
6	progressbar.bar module	29
7	progressbar.base module	39
8	progressbar.utils module	41
9	progressbar.widgets module	45
10	History	55
11	Text progress bar library for Python.	57
11.1	Install	57
11.2	Introduction	57
11.3	Security contact information	58
11.4	Known issues	58
11.5	Links	59
11.6	Usage	59

11.6.1	Wrapping an iterable	59
11.6.2	Progressbars with logging	59
11.6.3	Multiple (threaded) progressbars	60
11.6.4	Context wrapper	60
11.6.5	Combining progressbars with print output	61
11.6.6	Progressbar with unknown length	61
11.6.7	Bar with custom widgets	61
11.6.8	Bar with wide Chinese (or other multibyte) characters	61
11.6.9	Showing multiple independent progress bars in parallel	62
11.7	Indices and tables	64
Python Module Index		65
Index		67

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.

1.1 Wrapping an iterable

```
import time
import progressbar

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

1.2 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)
```

1.3 Combining progressbars with print output

```
import time
import progressbar

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

1.4 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)
```

1.5 Bar with custom widgets

```
import time
import progressbar

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

EXAMPLES

```
#!/usr/bin/python
from __future__ import annotations

import contextlib
import functools
import os
import random
import sys
import time
import typing

import progressbar

examples: list[typing.Callable[[typing.Any], typing.Any]] = []

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)
    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)
```

(continues on next page)

(continued from previous page)

```

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

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

@example
def parallelBars_multibar_example():
    if os.name == 'nt':
        print('Skipping multibar example on Windows due to threading '
              'incompatibilities with the example code.')
        return

    BARS = 5
    N = 50

    def do_something(bar):
        for _ in bar(range(N)):
            # Sleep up to 0.1 seconds
            time.sleep(random.random() * 0.1)

    with progressbar.MultiBar() as multibar:
        bar_labels = []
        for i in range(BARS):
            # Get a progressbar
            bar_label = 'Bar #%d' % i
            bar_labels.append(bar_label)
            multibar[bar_label]

            for _ in range(N * BARS):
                time.sleep(0.005)

                bar_i = random.randrange(0, BARS)
                bar_label = bar_labels[bar_i]
                # Increment one of the progress bars at random
                multibar[bar_label].increment()

@example
def multipleBars_line_offset_example():
    BARS = 5
    N = 100

    bars = [

```

(continues on next page)

(continued from previous page)

```

    progressbar.ProgressBar(
        max_value=N,
        # We add 1 to the line offset to account for the `print_fd`
        line_offset=i + 1,
        max_error=False,
    )
    for i in range(BARS)
]
# Create a file descriptor for regular printing as well
print_fd = progressbar.LineOffsetStreamWrapper(lines=0, stream=sys.stdout)
assert print_fd

# The progress bar updates, normally you would do something useful here
for _ in range(N * BARS):
    time.sleep(0.005)

    # Increment one of the progress bars at random
    bars[random.randrange(0, BARS)].increment()

# Cleanup the bars
for bar in bars:
    bar.finish()
    # Add a newline to make sure the next print starts on a new line
    print()

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

@example
def job_status_example():
    with progressbar.ProgressBar(
        redirect_stdout=True,
        widgets=[progressbar.widgets.JobStatusBar('status')],
    ) as bar:
        for _ in range(30):
            print('random', random.random())
            # Roughly 1/3 probability for each status ;)
            # Yes... probability is confusing at times
            if random.random() > 0.66:
                bar.increment(status=True)
            elif random.random() > 0.5:
                bar.increment(status=False)
            else:
                bar.increment(status=None)
            time.sleep(0.1)

@example

```

(continues on next page)

(continued from previous page)

```
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'),
    ]
    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
```

(continues on next page)

(continued from previous page)

```

        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 _ 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)

```

@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 _ 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 = [

```

(continues on next page)

(continued from previous page)

```

        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 granular_progress_example():
    widgets = [
        progressbar.GranularBar(markers=' ', left='', right='|'),
        progressbar.GranularBar(markers=' ', left='', right='|'),
        progressbar.GranularBar(markers=' ', left='', right='|'),
        progressbar.GranularBar(markers=' ', left='', right='|'),
        progressbar.GranularBar(markers=' ', left='', right='|'),
        progressbar.GranularBar(markers=' .o0', left='', right=''),
    ]
    for _ in progressbar.progressbar(list(range(100)), widgets=widgets):
        time.sleep(0.03)

    for _ in progressbar.progressbar(iter(range(100)), widgets=widgets):
        time.sleep(0.03)

@example
def percentage_label_bar_example():
    widgets = [progressbar.PercentageLabelBar()]
    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 file_transfer_example():
    widgets = [
        'Test: ',
        progressbar.Percentage(),
        ' ',
        progressbar.Bar(marker=progressbar.RotatingMarker()),
        ' ',
        progressbar.ETA(),
        ' ',
    ]

```

(continues on next page)

(continued from previous page)

```

        progressbar.FileTransferSpeed(),
    ]
    bar = progressbar.ProgressBar(widgets=widgets, max_value=1000).start()
    for i in range(100):
        # do something
        time.sleep(0.01)
        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:
                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
        time.sleep(0.01)
        bar.update(5 * i + 1)
    bar.finish()

```

@example

```

def double_bar_example():
    widgets = [
        progressbar.Bar('>'),
        ' ',
        progressbar.ETA(),
        ' ',
        progressbar.ReverseBar('<'),
    ]

```

(continues on next page)

(continued from previous page)

```

]
bar = progressbar.ProgressBar(widgets=widgets, max_value=1000).start()
for i in range(100):
    # do something
    time.sleep(0.01)
    bar.update(10 * i + 1)
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):
        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

(continues on next page)

(continued from previous page)

```

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

@example
def animated_marker():
    bar = progressbar.ProgressBar(
        widgets=['Working: ', progressbar.AnimatedMarker()]
    )
    for _ 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 _ 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 _ in bar(i for i in range(15)):
        time.sleep(0.1)

@example

```

(continues on next page)

(continued from previous page)

```
def format_label():
    widgets = [
        progressbar.FormatLabel('Processed: %(value)d lines (in: %(elapsed)s)')
    ]
    bar = progressbar.ProgressBar(widgets=widgets)
    for _ 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 _ 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 _ 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 _ 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 _ in bar(i for i in range(24)):
            time.sleep(0.1)
    except UnicodeError:
        sys.stdout.write('Unicode error: skipping example')
```

(continues on next page)

(continued from previous page)

```

@example
def format_label_bouncer():
    widgets = [
        progressbar.FormatLabel('Bouncer: value %(value)d - '),
        progressbar.BouncingBar(),
    ]
    bar = progressbar.ProgressBar(widgets=widgets)
    for _ 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 _ 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, contextlib.suppress(
        ValueError):
        progress.update(2)

@example
def reaching_maximum():
    progress = progressbar.ProgressBar(max_value=1)
    with contextlib.suppress(RuntimeError):
        progress.update(1)

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

```

(continues on next page)

(continued from previous page)

```

        progress.update(0)

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

@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 _ in range(10):
        # do something
        time.sleep(0.1)
        bar += 1
    bar.finish()

@example
def increment_bar_with_output_redirection():
    widgets = [

```

(continues on next page)

(continued from previous page)

```

    '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)
bar.finish()

```

@example

```

def eta_types_demonstration():
    widgets = [
        progressbar.Percentage(),
        ' ETA: ',
        progressbar.ETA(),
        ' Adaptive : ',
        progressbar.AdaptiveETA(),
        ' Smoothing(a=0.1): ',
        progressbar.SmoothingETA(smoothing_parameters=dict(alpha=0.1)),
        ' Smoothing(a=0.9): ',
        progressbar.SmoothingETA(smoothing_parameters=dict(alpha=0.9)),
        ' Absolute: ',
        progressbar.AbsoluteETA(),
        ' Transfer: ',
        progressbar.FileTransferSpeed(),
        ' Adaptive T: ',
        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()

```

(continues on next page)

(continued from previous page)

```
@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 _ 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 _ in bar(iter(range(100)), 100):
        # iter range is a way to get an iterator in both python 2 and 3
        time.sleep(0.01)
```

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

(continues on next page)

(continued from previous page)

```

# 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': '--'},
        max_value=10 * num_subtasks,
    ) as bar:
        for tasks_name, subtasks in tasks.items():
            for subtask_name in subtasks:
                for _ in range(10):
                    bar.update(
                        bar.value + 1, task=tasks_name, subtask=subtask_name

```

(continues on next page)

(continued from previous page)

```

        )
        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 _ in bar(range(200)):
        time.sleep(0.02)

@example
def eta_on_generators():
    def gen():
        for _ in range(200):
            yield None

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

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

```

(continues on next page)

(continued from previous page)

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

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

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

def test(*tests):
    if tests:
        no_tests = True
        for example in examples:
            for test in tests:
                if test in example.__name__:
                    example()
                    no_tests = False
                    break

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

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


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:

3.1 Types of Contributions

3.1.1 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.

3.1.2 Fix Bugs

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

3.1.3 Implement Features

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

3.1.4 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.

3.1.5 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 :)

3.2 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.

3.3 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.

3.4 Tips

To run a subset of tests:

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


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
```


PROGRESSBAR.SHORTCUTS MODULE

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

PROGRESSBAR.BAR MODULE

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

Bases: `ABC`

```
term_width: int = 80
```

The terminal width. This should be automatically detected but will fall back to 80 if auto detection is not possible.

```
widgets: MutableSequence[WidgetBase | str]
```

The widgets to render, defaults to the result of `default_widget()`

```
max_error: bool
```

When going beyond the `max_value`, raise an error if `True` or silently ignore otherwise

```
prefix: str | None
```

Prefix the progressbar with the given string

```
suffix: str | None
```

Suffix the progressbar with the given string

```
left_justify: bool
```

Justify to the left if `True` or the right if `False`

```
widget_kwargs: Dict[str, Any]
```

The default keyword arguments for the `default_widgets` if no widgets are configured

```
custom_len: Callable[[str], int]
```

```
initial_start_time: datetime | None
```

The time the progress bar was started

```
poll_interval: float | None
```

The interval to poll for updates in seconds if there are updates

```
min_poll_interval: float
```

The minimum interval to poll for updates in seconds even if there are no updates

```
num_intervals: int = 0
```

The number of intervals that can fit on the screen with a minimum of 100

Type

Deprecated

next_update: `int` = 0

The *next_update* is kept for compatibility with external libs: <https://github.com/WoLpH/python-progressbar/issues/207>

Type

Deprecated

value: `float`

Current progress ($\text{min_value} \leq \text{value} \leq \text{max_value}$)

previous_value: `float` | `None`

Previous progress value

min_value: `float`

The minimum/start value for the progress bar

max_value: `float` | `~typing.Type[<class 'progressbar.base.UnknownLength'>]`

Maximum (and final) value. Beyond this value an error will be raised unless the *max_error* parameter is *False*.

end_time: `datetime` | `None`

The time the progressbar reached *max_value* or when *finish()* was called.

start_time: `datetime` | `None`

The time *start()* was called or iteration started.

seconds_elapsed: `float`

Seconds between *start_time* and last call to *update()*

extra: `Dict[str, Any]`

Extra data for widgets with persistent state. This is used by sampling widgets for example. Since widgets can be shared between multiple progressbars we need to store the state with the progressbar.

get_last_update_time() → `datetime` | `None`

set_last_update_time(value: `datetime` | `None`)

property last_update_time: `datetime` | `None`

start(kwargs)**

update(value=None)

finish()

data() → `Dict[str, Any]`

started() → `bool`

finished() → `bool`

class progressbar.bar.ProgressBarBase(**kwargs)

Bases: `Iterable`, `ProgressBarMixinBase`

label: `str` = ''

index: `int` = -1

```
class progressbar.bar.DefaultFdMixin(fd: ~typing.TextIO = <_io.TextIOWrapper name='<stderr>'
                                     mode='w' encoding='utf-8'>, is_terminal: bool | None = None,
                                     line_breaks: bool | None = None, enable_colors:
                                     ~progressbar.env.ColorSupport | None = None, line_offset: int = 0,
                                     **kwargs)
```

Bases: *ProgressBarMixinBase*

fd: *TextIO* = <_io.TextIOWrapper name='<stderr>' mode='w' encoding='utf-8'>

is_ansi_terminal: *bool* | *None* = *False*

Set the terminal to be ANSI compatible. If a terminal is ANSI compatible we will automatically enable *colors* and disable *line_breaks*.

is_terminal: *bool* | *None*

Whether the file descriptor is a terminal or not. This is used to determine whether to use ANSI escape codes or not.

line_breaks: *bool* | *None* = *True*

Whether to print line breaks. This is useful for logging the progressbar. When disabled the current line is overwritten.

enable_colors: *ColorSupport* = *0*

Specify the type and number of colors to support. Defaults to auto detection based on the file descriptor type (i.e. interactive terminal) environment variables such as *COLORTERM* and *TERM*. Color output can be forced in non-interactive terminals using the *PROGRESSBAR_ENABLE_COLORS* environment variable which can also be used to force a specific number of colors by specifying *24bit*, *256* or *16*. For true (24 bit/16M) color support you can use *COLORTERM=truecolor*. For 256 color support you can use *TERM=xterm-256color*. For 16 colorsupport you can use *TERM=xterm*.

print(*args: *Any*, **kwargs: *Any*) → *None*

start(**kwargs)

update(*args: *Any*, **kwargs: *Any*) → *None*

finish(*args: *Any*, **kwargs: *Any*) → *None*

```
class progressbar.bar.ResizableMixin(term_width: int | None = None, **kwargs)
```

Bases: *ProgressBarMixinBase*

finish()

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

Bases: *DefaultFdMixin*

redirect_stderr: *bool* = *False*

redirect_stdout: *bool* = *False*

stdout: *WrappingIO* | *IO*

stderr: *WrappingIO* | *IO*

start(*args, **kwargs)

update(value: *float* | *None* = *None*)

```
finish(end='\n')
```

```
class progressbar.bar.ProgressBar(min_value: float = 0, max_value: float | ~typing.Type[<class
    'progressbar.base.UnknownLength'>] | None = None, widgets:
    ~typing.Sequence[~progressbar.widgets.WidgetBase | str] | None =
    None, left_justify: bool = True, initial_value: float = 0, poll_interval:
    float | None = None, widget_kwargs: ~typing.Dict[str, ~typing.Any] |
    None = None, custom_len: ~typing.Callable[[str], int] = <function
    len_color>, max_error=True, prefix=None, suffix=None,
    variables=None, min_poll_interval=None, **kwargs)
```

Bases: [StdRedirectMixin](#), [ResizableMixin](#), [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
- **line_offset** (*int*) – The number of lines to offset the progressbar from your current line. This is useful if you have other output or multiple progressbars

A common way of using it is like:

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

(continues on next page)

```
...
>>> progress.finish()
```

initial_start_time: `datetime` | `None`

The time the progress bar was started

poll_interval: `float` | `None`

The interval to poll for updates in seconds if there are updates

min_poll_interval: `float`

The minimum interval to poll for updates in seconds even if there are no updates

property `dynamic_messages`

init()

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

property `percentage:` `float` | `None`

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
>>> 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
```

data() → `Dict[str, Any]`

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()**next()****increment**(*value=1*, **args*, ***kwargs*)**update**(*value=None*, *force=False*, ***kwargs*)

Updates the ProgressBar to a new value.

stdout: *WrappingIO* | *IO***stderr**: *WrappingIO* | *IO***is_terminal**: *bool* | *None*

Whether the file descriptor is a terminal or not. This is used to determine whether to use ANSI escape codes or not.

previous_value: *float* | *None*

Previous progress value

end_time: *datetime* | *None*The time the progressbar reached *max_value* or when *finish()* was called.**start_time**: *datetime* | *None*The time *start()* was called or iteration started.**seconds_elapsed**: *float*Seconds between *start_time* and last call to *update()***extra**: *Dict[str, Any]*

Extra data for widgets with persistent state. This is used by sampling widgets for example. Since widgets can be shared between multiple progressbars we need to store the state with the progressbar.

start(*max_value=None*, *init=True*, **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
- **init** (*bool*) – (Re)Initialize the progressbar, this is useful if you wish to reuse the same progressbar but can be disabled if data needs to be persisted between runs

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

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

property currval

Legacy method to make progressbar-2 compatible with the original progressbar package.

```
class progressbar.bar.DataTransferBar(min_value: float = 0, max_value: float | ~typing.Type[<class
'progressbar.base.UnknownLength'>] | None = None, widgets:
~typing.Sequence[~progressbar.widgets.WidgetBase | str] | None = None,
left_justify: bool = True, initial_value: float = 0,
poll_interval: float | None = None, widget_kwargs:
~typing.Dict[str, ~typing.Any] | None = None, custom_len:
~typing.Callable[[str], int] = <function len_color>,
max_error=True, prefix=None, suffix=None, variables=None,
min_poll_interval=None, **kwargs)
```

Bases: [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.NullBar(min_value: float = 0, max_value: float | ~typing.Type[<class
'progressbar.base.UnknownLength'>] | None = None, widgets:
~typing.Sequence[~progressbar.widgets.WidgetBase | str] | None = None,
left_justify: bool = True, initial_value: float = 0, poll_interval: float | None =
None, widget_kwargs: ~typing.Dict[str, ~typing.Any] | None = None,
custom_len: ~typing.Callable[[str], int] = <function len_color>,
max_error=True, prefix=None, suffix=None, variables=None,
min_poll_interval=None, **kwargs)
```

Bases: [ProgressBar](#)

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

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
- **init** (*bool*) – (Re)Initialize the progressbar, this is useful if you wish to reuse the same progressbar but can be disabled if data needs to be persisted between runs

```
>>> 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.

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

PROGRESSBAR.BASE MODULE

```
class progressbar.base.FalseMeta
    Bases: type
class progressbar.base.Undefined
    Bases: object
class progressbar.base.UnknownLength
    Bases: object
```


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
```

(continues on next page)

(continued from previous page)

```
Traceback (most recent call last):  
...  
AttributeError: No such attribute: spam
```

```
class progressbar.utils.StreamWrapper
```

```
    Bases: object
```

```
    Wrap stdout and stderr globally.
```

```
    capturing: int = 0
```

```
    excepthook(exc_type, exc_value, exc_traceback)
```

```
    flush() → None
```

```
    listeners: set
```

```
    needs_clear() → bool
```

```
    original_excepthook: Callable[[Type[BaseException], BaseException, TracebackType |  
    None], None]
```

```
    start_capturing(bar: ProgressBarMixinBase | None = None) → None
```

```
    stderr: TextIO | WrappingIO
```

```
    stdout: TextIO | WrappingIO
```

```
    stop_capturing(bar: ProgressBarMixinBase | None = None) → None
```

```
    unwrap(stdout: bool = False, stderr: bool = False) → None
```

```
    unwrap_excepthook() → None
```

```
    unwrap_stderr() → None
```

```
    unwrap_stdout() → None
```

```
    update_capturing() → None
```

```
    wrap(stdout: bool = False, stderr: bool = False) → None
```

```
    wrap_excepthook() → None
```

```
    wrap_stderr() → WrappingIO
```

```
    wrap_stdout() → WrappingIO
```

```
    wrapped_excepthook: int = 0
```

```
    wrapped_stderr: int = 0
```

```
    wrapped_stdout: int = 0
```

```
class progressbar.utils.WrappingIO(target: base.IO, capturing: bool = False, listeners:  
    types.Optional[types.Set[ProgressBar]] = None)
```

```
    Bases: object
```

```

buffer: StringIO
capturing: bool
close() → None
fileno() → int
flush() → None
flush_target() → None
isatty() → bool
listeners: set
needs_clear: bool = False
read(n: int = -1) → str
readable() → bool
readline(limit: int = -1) → str
readlines(hint: int = -1) → list[str]
seek(offset: int, whence: int = 0) → int
seekable() → bool
target: IO
tell() → int
truncate(size: int | None = None) → int
writable() → bool
write(value: str) → int
writelines(lines: Iterable[str]) → None

```

`progressbar.utils.deltas_to_seconds(*deltas, default: ~typing.Type[ValueError] | None = <class 'ValueError'>) → int | float | None`

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):

```

(continues on next page)

(continued from previous page)

```
...
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.len_color(value: str | bytes) → int`

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: StringT) → StringT`

Return the *value* without ANSI escape codes.

```
>>> no_color(b'[1234]abc')
b'abc'
>>> str(no_color(u'[1234]abc'))
'abc'
>>> str(no_color('[1234]abc'))
'abc'
>>> no_color(123)
Traceback (most recent call last):
...
TypeError: `value` must be a string or bytes, got 123
```

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: [ETA](#)

Widget which attempts to estimate the absolute time of arrival.

```
class progressbar.widgets.AdaptiveETA(exponential_smoothing=True, exponential_smoothing_factor=0.1,
                                     **kwargs)
```

Bases: [ETA](#), [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.

exponential_smoothing: **bool**

exponential_smoothing_factor: **float**

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

Bases: [FileTransferSpeed](#), [SamplesMixin](#)

Widget 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: [TimeSensitiveWidgetBase](#)

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

```
class progressbar.widgets.AutoWidthWidgetBase(*args, fixed_colors=None, gradient_colors=None,
                                     **kwargs)
```

Bases: [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: [AutoWidthWidgetBase](#)

A progress bar which stretches to fill the line.

bg: Color | ColorGradient | None = None

fg: Color | ColorGradient | None = <progressbar.terminal.base.ColorGradient object>

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

Bases: Bar, TimeSensitiveWidgetBase

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

INTERVAL = datetime.timedelta(microseconds=1000000)

class progressbar.widgets.ColoredMixin

Bases: object

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

Bases: FormatWidgetMixin, WidgetBase

Displays the current count.

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

Bases: FormatWidgetMixin, 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: FormatWidgetMixin, 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: 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: 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: FormatWidgetMixin, TimeSensitiveWidgetBase

Widget for showing the current transfer speed (useful for file transfers).

```
class progressbar.widgets.FormatCustomText(format: str, mapping: Dict[str, Any] | None = None,
                                           **kwargs)
```

Bases: `FormatWidgetMixin`, `WidgetBase`

`copy = False`

`mapping: Dict[str, Any] = {}`

`update_mapping(**mapping: Dict[str, Any])`

```
class progressbar.widgets.FormatLabel(format: str, **kwargs)
```

Bases: `FormatWidgetMixin`, `WidgetBase`

Displays a formatted label.

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

```
mapping: ClassVar[Dict[str, Tuple[str, Any]]] = {'elapsed':
('total_seconds_elapsed', <function format_time>), 'finished': ('end_time', None),
'last_update': ('last_update_time', None), 'max': ('max_value', None), 'seconds':
('seconds_elapsed', None), 'start': ('start_time', None), 'value': ('value',
None)}
```

```
class progressbar.widgets.FormatLabelBar(format, **kwargs)
```

Bases: `FormatLabel`, `Bar`

A bar which has a formatted label in the center.

```
class progressbar.widgets.FormatWidgetMixin(format: str, new_style: bool = False, **kwargs)
```

Bases: `ABC`

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

```
get_format(progress: ProgressBarMixinBase, data: Data, format: types.Optional[str] = None) → str
```

```
class progressbar.widgets.GranularBar(markers=' ', left='|', right='|', **kwargs)
```

Bases: [AutoWidthWidgetBase](#)

A progressbar that can display progress at a sub-character granularity by using multiple marker characters.

Examples of markers:

- Smooth: `` (default)
- Bar: ``
- Snake: ``
- Fade in: ``
- Dots: ``
- Growing circles: ``.oO`

The markers can be accessed through `GranularMarkers`. `GranularMarkers.dots` for example

```
class progressbar.widgets.GranularMarkers
```

Bases: `object`

```
bar = ' '
```

```
dots = ' '
```

```
fade_in = ' '
```

```
growing_circles = ' .oO'
```

```
smooth = ' '
```

```
snake = ' '
```

```
class progressbar.widgets.JobStatusBar(name: str, left='|', right='|', fill=' ', fill_left=True,
                                         success_fg_color=((0, 128, 0), (120, 100, 25), 'Green', 2),
                                         success_bg_color=None, success_marker="",
                                         failure_fg_color=((255, 0, 0), (0, 100, 50), 'Red', 9),
                                         failure_bg_color=None, failure_marker='X', **kwargs)
```

Bases: [Bar](#), [VariableMixin](#)

Widget which displays the job status as markers on the bar.

The status updates can be given either as a boolean or as a string. If it's a string, it will be displayed as-is. If it's a boolean, it will be displayed as a marker (default: " for success, 'X' for failure) configurable through the `success_marker` and `failure_marker` parameters.

Parameters

- **name** – The name of the variable to use for the status updates.
- **left** – The left border of the bar.
- **right** – The right border of the bar.
- **fill** – The fill character of the bar.
- **fill_left** – Whether to fill the bar from the left or the right.
- **success_fg_color** – The foreground color to use for successful jobs.
- **success_bg_color** – The background color to use for successful jobs.
- **success_marker** – The marker to use for successful jobs.

- **failure_fg_color** – The foreground color to use for failed jobs.
- **failure_bg_color** – The background color to use for failed jobs.
- **failure_marker** – The marker to use for failed jobs.

failure_bg_color: Color | None = None

failure_fg_color: Color | None = ((255, 0, 0), (0, 100, 50), 'Red', 9)

failure_marker: str = 'X'

job_markers: list[str]

success_bg_color: Color | None = None

success_fg_color: Color | None = ((0, 128, 0), (120, 100, 25), 'Green', 2)

success_marker: str = ''

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

Bases: [MultiRangeBar](#)

get_values(progress: [ProgressBarMixinBase](#), data: Data)

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

Bases: [Bar](#), [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],
    ...
]
```

get_values(progress: [ProgressBarMixinBase](#), data: Data)

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

Bases: [FormatWidgetMixin](#), [ColoredMixin](#), [WidgetBase](#)

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

get_format(progress: [ProgressBarMixinBase](#), data: Data, format=None)

class progressbar.widgets.**PercentageLabelBar**(format='%(percentage)2d%%', na='N/A%%', **kwargs)

Bases: [Percentage](#), [FormatLabelBar](#)

A bar which displays the current percentage in the center.

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

Bases: [Bar](#)

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

progressbar.widgets.RotatingMarker

alias of [AnimatedMarker](#)

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

Bases: [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
SliceableDeque([1, 1])
```

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

get_sample_times(progress: [ProgressBarMixinBase](#), data: [Data](#))

get_sample_values(progress: [ProgressBarMixinBase](#), data: [Data](#))

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

Bases: [FormatWidgetMixin](#), [ColoredMixin](#), [WidgetBase](#)

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

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

max_width_cache: dict[str | tuple[float, float | ~typing.Type[<class
'progressbar.base.UnknownLength'>]], int | None]

```
class progressbar.widgets.SmoothingETA(smoothing_algorithm:
                                       type[~progressbar.algorithms.SmoothingAlgorithm] = <class
                                       'progressbar.algorithms.ExponentialMovingAverage'>,
                                       smoothing_parameters: dict[str, float] | None = None, **kwargs)
```

Bases: [ETA](#)

WidgetBase which attempts to estimate the time of arrival using an exponential moving average (EMA) of the speed.

EMA applies more weight to recent data points and less to older ones, and doesn’t require storing all past values. This approach works well with varying data points and smooths out fluctuations effectively.

smoothing_algorithm: `SmoothingAlgorithm`

smoothing_parameters: `dict[str, float]`

class `progressbar.widgets.TFixedColors`

Bases: `TypedDict`

bg_none: `Color | None`

fg_none: `Color | None`

class `progressbar.widgets.TGradientColors`

Bases: `TypedDict`

bg: `Color | ColorGradient | None`

fg: `Color | ColorGradient | None`

class `progressbar.widgets.TimeSensitiveWidgetBase(*args, fixed_colors=None, gradient_colors=None, **kwargs)`

Bases: `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: `FormatLabel, TimeSensitiveWidgetBase`

WidgetBase which displays the elapsed seconds.

static format_time(*timestamp: timedelta | date | datetime | str | int | float | None, precision: timedelta = datetime.timedelta(seconds=1)*) → `str`

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)
Traceback (most recent call last):
...
TypeError: Unknown type ...
```

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

Bases: `FormatWidgetMixin`, `VariableMixin`, `WidgetBase`

Displays a custom variable.

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

Bases: `object`

Mixin to display a custom user variable.

```
class progressbar.widgets.WidgetBase(*args, fixed_colors=None, gradient_colors=None, **kwargs)
```

Bases: `WidthWidgetMixin`

The base class for all widgets.

The ProgressBar will call the widget's update value when the widget should be updated. The widget's size may change between calls, but the widget may display incorrectly if the size changes drastically and repeatedly.

The INTERVAL timedelta informs the ProgressBar that it should be updated more often because it is time sensitive.

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

WARNING: Widgets can be shared between multiple progressbars so any state information specific to a progressbar should be stored within the progressbar instead of the widget.

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
- `weight`: Widgets with a higher `weight` will be calculated before widgets with a lower one
- `copy`: Copy this widget when initializing the progress bar so the progressbar can be reused. Some widgets such as the `FormatCustomText` require the shared state so this needs to be optional

`copy = True`

`property uses_colors`

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

Bases: `ABC`

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:
...     term_width = 0
```

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

(continues on next page)

(continued from previous page)

```
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*: [ProgressBarMixinBase](#))

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.

HISTORY

releases or the commit log:

- <https://github.com/WoLpH/python-progressbar/releases>
- <https://github.com/WoLpH/python-progressbar/commits/develop>

Hint: click on the ... button to see the change message.

TEXT PROGRESS BAR LIBRARY FOR PYTHON.

Build status:

Coverage:

11.1 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
```

11.2 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 is based on the old Python progressbar package that was published on the now defunct Google Code. Since that project was completely abandoned by its developer and the developer did not respond to email, I decided to fork the package. This package is still backwards compatible with the original progressbar package so you can safely use it as a drop-in replacement for existing project.

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`
- `FormatLabelBar`
- `GranularBar`
- `Percentage`
- `PercentageLabelBar`
- `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.

11.3 Security contact information

To report a security vulnerability, please use the [Tidelift security contact](#). Tidelift will coordinate the fix and disclosure.

11.4 Known issues

- The JetBrains (PyCharm, etc) editors work out of the box, but for more advanced features such as the *MultiBar* support you will need to enable the “Enable terminal in output console” checkbox in the Run dialog.
- The IDLE editor doesn’t support these types of progress bars at all: <https://bugs.python.org/issue23220>
- 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>

11.5 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/>

11.6 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.

11.6.1 Wrapping an iterable

```
import time
import progressbar

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

11.6.2 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()

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

11.6.3 Multiple (threaded) progressbars

```
import random
import threading
import time

import progressbar

BARS = 5
N = 50

def do_something(bar):
    for i in bar(range(N)):
        # Sleep up to 0.1 seconds
        time.sleep(random.random() * 0.1)

        # print messages at random intervals to show how extra output works
        if random.random() > 0.9:
            bar.print('random message for bar', bar, i)

with progressbar.MultiBar() as multibar:
    for i in range(BARS):
        # Get a progressbar
        bar = multibar[f'Thread label here {i}']
        # Create a thread and pass the progressbar
        threading.Thread(target=do_something, args=(bar,)).start()
```

11.6.4 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)
```

11.6.5 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)
```

11.6.6 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)
```

11.6.7 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)
```

11.6.8 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:
```

(continues on next page)

(continued from previous page)

```

        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)

```

11.6.9 Showing multiple independent progress bars in parallel

```

import random
import sys
import time

import progressbar

BARS = 5
N = 100

# Construct the list of progress bars with the `line_offset` so they draw
# below each other
bars = []
for i in range(BARS):
    bars.append(
        progressbar.ProgressBar(
            max_value=N,
            # We add 1 to the line offset to account for the `print_fd`
            line_offset=i + 1,
            max_error=False,
        )
    )

# Create a file descriptor for regular printing as well
print_fd = progressbar.LineOffsetStreamWrapper(lines=0, stream=sys.stdout)

# The progress bar updates, normally you would do something useful here
for i in range(N * BARS):
    time.sleep(0.005)

    # Increment one of the progress bars at random
    bars[random.randrange(0, BARS)].increment()

```

(continues on next page)

(continued from previous page)

```
# Print a status message to the `print_fd` below the progress bars
print(f'Hi, we are at update {i+1} of {N * BARS}', file=print_fd)

# Cleanup the bars
for bar in bars:
    bar.finish()

# Add a newline to make sure the next print starts on a new line
print()
```

Naturally we can do this from separate threads as well:

```
import random
import threading
import time

import progressbar

BARS = 5
N = 100

# Create the bars with the given line offset
bars = []
for line_offset in range(BARS):
    bars.append(progressbar.ProgressBar(line_offset=line_offset, max_value=N))

class Worker(threading.Thread):
    def __init__(self, bar):
        super().__init__()
        self.bar = bar

    def run(self):
        for i in range(N):
            time.sleep(random.random() / 25)
            self.bar.update(i)

for bar in bars:
    Worker(bar).start()

print()
```

11.7 Indices and tables

- [genindex](#)
- [modindex](#)
- [search](#)

PYTHON MODULE INDEX

p

- `progressbar.bar`, [29](#)
- `progressbar.base`, [39](#)
- `progressbar.shortcuts`, [27](#)
- `progressbar.utils`, [41](#)
- `progressbar.widgets`, [45](#)

A

AbsoluteETA (class in *progressbar.widgets*), 45
 AdaptiveETA (class in *progressbar.widgets*), 45
 AdaptiveTransferSpeed (class in *progressbar.widgets*), 45
 AnimatedMarker (class in *progressbar.widgets*), 45
 AttributeDict (class in *progressbar.utils*), 41
 AutoWidthWidgetBase (class in *progressbar.widgets*), 45

B

Bar (class in *progressbar.widgets*), 45
 bar (*progressbar.widgets.GranularMarkers* attribute), 48
 bg (*progressbar.widgets.Bar* attribute), 45
 bg (*progressbar.widgets.TGradientColors* attribute), 51
 bg_none (*progressbar.widgets.TFixedColors* attribute), 51
 BouncingBar (class in *progressbar.widgets*), 46
 buffer (*progressbar.utils.WrappingIO* attribute), 42

C

capturing (*progressbar.utils.StreamWrapper* attribute), 42
 capturing (*progressbar.utils.WrappingIO* attribute), 43
 check_size() (*progressbar.widgets.WidthWidgetMixin* method), 53
 close() (*progressbar.utils.WrappingIO* method), 43
 ColoredMixin (class in *progressbar.widgets*), 46
 copy (*progressbar.widgets.FormatCustomText* attribute), 47
 copy (*progressbar.widgets.WidgetBase* attribute), 52
 Counter (class in *progressbar.widgets*), 46
 create_marker() (in module *progressbar.widgets*), 53
 create_wrapper() (in module *progressbar.widgets*), 53
 current_datetime() (*progressbar.widgets.CurrentTime* method), 46
 current_time() (*progressbar.widgets.CurrentTime* method), 46
 CurrentTime (class in *progressbar.widgets*), 46
 currval (*progressbar.bar.ProgressBar* property), 36
 custom_len (*progressbar.bar.ProgressBar* attribute), 33

custom_len (*progressbar.bar.ProgressBarMixinBase* attribute), 29

D

data() (*progressbar.bar.ProgressBar* method), 34
 data() (*progressbar.bar.ProgressBarMixinBase* method), 30
 DataSize (class in *progressbar.widgets*), 46
 DataTransferBar (class in *progressbar.bar*), 36
 DEFAULT_FORMAT (*progressbar.widgets.SimpleProgress* attribute), 50
 default_widgets() (*progressbar.bar.DataTransferBar* method), 36
 default_widgets() (*progressbar.bar.ProgressBar* method), 35
 DefaultFdMixin (class in *progressbar.bar*), 30
 deltas_to_seconds() (in module *progressbar.utils*), 43
 dots (*progressbar.widgets.GranularMarkers* attribute), 48
 dynamic_messages (*progressbar.bar.ProgressBar* property), 34
 DynamicMessage (class in *progressbar.widgets*), 46

E

enable_colors (*progressbar.bar.DefaultFdMixin* attribute), 31
 end_time (*progressbar.bar.ProgressBar* attribute), 35
 end_time (*progressbar.bar.ProgressBarMixinBase* attribute), 30
 ETA (class in *progressbar.widgets*), 46
 excepthook() (*progressbar.utils.StreamWrapper* method), 42
 exponential_smoothing (*progressbar.widgets.AdaptiveETA* attribute), 45
 exponential_smoothing_factor (*progressbar.widgets.AdaptiveETA* attribute), 45
 extra (*progressbar.bar.ProgressBar* attribute), 35
 extra (*progressbar.bar.ProgressBarMixinBase* attribute), 30

F

`fade_in` (*progressbar.widgets.GranularMarkers* attribute), 48

`failure_bg_color` (*progressbar.widgets.JobStatusBar* attribute), 49

`failure_fg_color` (*progressbar.widgets.JobStatusBar* attribute), 49

`failure_marker` (*progressbar.widgets.JobStatusBar* attribute), 49

`FalseMeta` (class in *progressbar.base*), 39

`fd` (*progressbar.bar.DefaultFdMixin* attribute), 31

`fg` (*progressbar.widgets.Bar* attribute), 46

`fg` (*progressbar.widgets.TGradientColors* attribute), 51

`fg_none` (*progressbar.widgets.TFixedColors* attribute), 51

`fileno()` (*progressbar.utils.WrappingIO* method), 43

`FileTransferSpeed` (class in *progressbar.widgets*), 46

`finish()` (*progressbar.bar.DefaultFdMixin* method), 31

`finish()` (*progressbar.bar.NullBar* method), 37

`finish()` (*progressbar.bar.ProgressBar* method), 36

`finish()` (*progressbar.bar.ProgressBarMixinBase* method), 30

`finish()` (*progressbar.bar.ResizableMixin* method), 31

`finish()` (*progressbar.bar.StdRedirectMixin* method), 31

`finished()` (*progressbar.bar.ProgressBarMixinBase* method), 30

`flush()` (*progressbar.utils.StreamWrapper* method), 42

`flush()` (*progressbar.utils.WrappingIO* method), 43

`flush_target()` (*progressbar.utils.WrappingIO* method), 43

`format_time()` (*progressbar.widgets.Timer* static method), 51

`FormatCustomText` (class in *progressbar.widgets*), 46

`FormatLabel` (class in *progressbar.widgets*), 47

`FormatLabelBar` (class in *progressbar.widgets*), 47

`FormatWidgetMixin` (class in *progressbar.widgets*), 47

G

`get_format()` (*progressbar.widgets.FormatWidgetMixin* method), 47

`get_format()` (*progressbar.widgets.Percentage* method), 49

`get_last_update_time()` (*progressbar.bar.ProgressBarMixinBase* method), 30

`get_sample_times()` (*progressbar.widgets.SamplesMixin* method), 50

`get_sample_values()` (*progressbar.widgets.SamplesMixin* method), 50

`get_values()` (*progressbar.widgets.MultiProgressBar* method), 49

`get_values()` (*progressbar.widgets.MultiRangeBar* method), 49

`GranularBar` (class in *progressbar.widgets*), 47

`GranularMarkers` (class in *progressbar.widgets*), 48

`growing_circles` (*progressbar.widgets.GranularMarkers* attribute), 48

I

`increment()` (*progressbar.bar.ProgressBar* method), 35

`index` (*progressbar.bar.ProgressBarBase* attribute), 30

`init()` (*progressbar.bar.ProgressBar* method), 34

`initial_start_time` (*progressbar.bar.ProgressBar* attribute), 33

`initial_start_time` (*progressbar.bar.ProgressBarMixinBase* attribute), 29

`INTERVAL` (*progressbar.widgets.BouncingBar* attribute), 46

`INTERVAL` (*progressbar.widgets.CurrentTime* attribute), 46

`INTERVAL` (*progressbar.widgets.TimeSensitiveWidgetBase* attribute), 51

`is_ansi_terminal` (*progressbar.bar.DefaultFdMixin* attribute), 31

`is_terminal` (*progressbar.bar.DefaultFdMixin* attribute), 31

`is_terminal` (*progressbar.bar.ProgressBar* attribute), 35

`isatty()` (*progressbar.utils.WrappingIO* method), 43

J

`job_markers` (*progressbar.widgets.JobStatusBar* attribute), 49

`JobStatusBar` (class in *progressbar.widgets*), 48

L

`label` (*progressbar.bar.ProgressBarBase* attribute), 30

`last_update_time` (*progressbar.bar.ProgressBarMixinBase* property), 30

`left_justify` (*progressbar.bar.ProgressBar* attribute), 33

`left_justify` (*progressbar.bar.ProgressBarMixinBase* attribute), 29

`len_color()` (in module *progressbar.utils*), 44

`line_breaks` (*progressbar.bar.DefaultFdMixin* attribute), 31

`listeners` (*progressbar.utils.StreamWrapper* attribute), 42

`listeners` (*progressbar.utils.WrappingIO* attribute), 43

M

`mapping` (*progressbar.widgets.FormatCustomText*

attribute), 47
 mapping (progressbar.widgets.FormatLabel attribute), 47
 max_error (progressbar.bar.ProgressBar attribute), 33
 max_error (progressbar.bar.ProgressBarMixinBase attribute), 29
 max_value (progressbar.bar.ProgressBar attribute), 33
 max_value (progressbar.bar.ProgressBarMixinBase attribute), 30
 max_width_cache (progressbar.widgets.SimpleProgress attribute), 50
 min_poll_interval (progressbar.bar.ProgressBar attribute), 34
 min_poll_interval (progressbar.bar.ProgressBarMixinBase attribute), 29
 min_value (progressbar.bar.ProgressBar attribute), 33
 min_value (progressbar.bar.ProgressBarMixinBase attribute), 30
 module
 progressbar.bar, 29
 progressbar.base, 39
 progressbar.shortcuts, 27
 progressbar.utils, 41
 progressbar.widgets, 45
 MultiProgressBar (class in progressbar.widgets), 49
 MultiRangeBar (class in progressbar.widgets), 49

N

needs_clear (progressbar.utils.WrappingIO attribute), 43
 needs_clear() (progressbar.utils.StreamWrapper method), 42
 next() (progressbar.bar.ProgressBar method), 35
 next_update (progressbar.bar.ProgressBarMixinBase attribute), 29
 no_color() (in module progressbar.utils), 44
 NullBar (class in progressbar.bar), 36
 num_intervals (progressbar.bar.ProgressBarMixinBase attribute), 29

O

original_excepthook (progressbar.utils.StreamWrapper attribute), 42

P

paused (progressbar.bar.ProgressBar attribute), 33
 Percentage (class in progressbar.widgets), 49
 percentage (progressbar.bar.ProgressBar property), 34
 PercentageLabelBar (class in progressbar.widgets), 49
 poll_interval (progressbar.bar.ProgressBar attribute), 34

poll_interval (progressbar.bar.ProgressBarMixinBase attribute), 29
 prefix (progressbar.bar.ProgressBar attribute), 33
 prefix (progressbar.bar.ProgressBarMixinBase attribute), 29
 previous_value (progressbar.bar.ProgressBar attribute), 35
 previous_value (progressbar.bar.ProgressBarMixinBase attribute), 30
 print() (progressbar.bar.DefaultFdMixin method), 31
 ProgressBar (class in progressbar.bar), 32
 progressbar() (in module progressbar.shortcuts), 27
 progressbar.bar module, 29
 progressbar.base module, 39
 progressbar.shortcuts module, 27
 progressbar.utils module, 41
 progressbar.widgets module, 45
 ProgressBarBase (class in progressbar.bar), 30
 ProgressBarMixinBase (class in progressbar.bar), 29

R

read() (progressbar.utils.WrappingIO method), 43
 readable() (progressbar.utils.WrappingIO method), 43
 readline() (progressbar.utils.WrappingIO method), 43
 readlines() (progressbar.utils.WrappingIO method), 43
 redirect_stderr (progressbar.bar.StdRedirectMixin attribute), 31
 redirect_stdout (progressbar.bar.StdRedirectMixin attribute), 31
 ResizableMixin (class in progressbar.bar), 31
 ReverseBar (class in progressbar.widgets), 49
 RotatingMarker (in module progressbar.widgets), 49

S

SamplesMixin (class in progressbar.widgets), 49
 seconds_elapsed (progressbar.bar.ProgressBar attribute), 35
 seconds_elapsed (progressbar.bar.ProgressBarMixinBase attribute), 30
 seek() (progressbar.utils.WrappingIO method), 43
 seekable() (progressbar.utils.WrappingIO method), 43
 set_last_update_time() (progressbar.bar.ProgressBarMixinBase method), 30
 SimpleProgress (class in progressbar.widgets), 50

smooth (*progressbar.widgets.GranularMarkers* attribute), 48

smoothing_algorithm (*progressbar.widgets.SmoothingETA* attribute), 50

smoothing_parameters (*progressbar.widgets.SmoothingETA* attribute), 51

SmoothingETA (class in *progressbar.widgets*), 50

snake (*progressbar.widgets.GranularMarkers* attribute), 48

start() (*progressbar.bar.DefaultFdMixin* method), 31

start() (*progressbar.bar.NullBar* method), 36

start() (*progressbar.bar.ProgressBar* method), 35

start() (*progressbar.bar.ProgressBarMixinBase* method), 30

start() (*progressbar.bar.StdRedirectMixin* method), 31

start_capturing() (*progressbar.utils.StreamWrapper* method), 42

start_time (*progressbar.bar.ProgressBar* attribute), 35

start_time (*progressbar.bar.ProgressBarMixinBase* attribute), 30

started() (*progressbar.bar.ProgressBarMixinBase* method), 30

stderr (*progressbar.bar.ProgressBar* attribute), 35

stderr (*progressbar.bar.StdRedirectMixin* attribute), 31

stderr (*progressbar.utils.StreamWrapper* attribute), 42

stdout (*progressbar.bar.ProgressBar* attribute), 35

stdout (*progressbar.bar.StdRedirectMixin* attribute), 31

stdout (*progressbar.utils.StreamWrapper* attribute), 42

StdRedirectMixin (class in *progressbar.bar*), 31

stop_capturing() (*progressbar.utils.StreamWrapper* method), 42

StreamWrapper (class in *progressbar.utils*), 42

string_or_lambda() (in module *progressbar.widgets*), 53

success_bg_color (*progressbar.widgets.JobStatusBar* attribute), 49

success_fg_color (*progressbar.widgets.JobStatusBar* attribute), 49

success_marker (*progressbar.widgets.JobStatusBar* attribute), 49

suffix (*progressbar.bar.ProgressBar* attribute), 33

suffix (*progressbar.bar.ProgressBarMixinBase* attribute), 29

T

target (*progressbar.utils.WrappingIO* attribute), 43

tell() (*progressbar.utils.WrappingIO* method), 43

term_width (*progressbar.bar.ProgressBarMixinBase* attribute), 29

TFixedColors (class in *progressbar.widgets*), 51

TGradientColors (class in *progressbar.widgets*), 51

Timer (class in *progressbar.widgets*), 51

TimeSensitiveWidgetBase (class in *progressbar.widgets*), 51

truncate() (*progressbar.utils.WrappingIO* method), 43

U

Undefined (class in *progressbar.base*), 39

UnknownLength (class in *progressbar.base*), 39

unwrap() (*progressbar.utils.StreamWrapper* method), 42

unwrap_excepthook() (*progressbar.utils.StreamWrapper* method), 42

unwrap_stderr() (*progressbar.utils.StreamWrapper* method), 42

unwrap_stdout() (*progressbar.utils.StreamWrapper* method), 42

update() (*progressbar.bar.DefaultFdMixin* method), 31

update() (*progressbar.bar.NullBar* method), 37

update() (*progressbar.bar.ProgressBar* method), 35

update() (*progressbar.bar.ProgressBarMixinBase* method), 30

update() (*progressbar.bar.StdRedirectMixin* method), 31

update_capturing() (*progressbar.utils.StreamWrapper* method), 42

update_mapping() (*progressbar.widgets.FormatCustomText* method), 47

uses_colors (*progressbar.widgets.WidgetBase* property), 52

V

value (*progressbar.bar.ProgressBar* attribute), 33

value (*progressbar.bar.ProgressBarMixinBase* attribute), 30

Variable (class in *progressbar.widgets*), 51

VariableMixin (class in *progressbar.widgets*), 52

W

widget_kwargs (*progressbar.bar.ProgressBar* attribute), 33

widget_kwargs (*progressbar.bar.ProgressBarMixinBase* attribute), 29

WidgetBase (class in *progressbar.widgets*), 52

widgets (*progressbar.bar.ProgressBar* attribute), 33

widgets (*progressbar.bar.ProgressBarMixinBase* attribute), 29

WidthWidgetMixin (class in *progressbar.widgets*), 52

wrap() (*progressbar.utils.StreamWrapper* method), 42

wrap_excepthook() (*progressbar.utils.StreamWrapper* method), 42

wrap_stderr() (*progressbar.utils.StreamWrapper* method), 42

wrap_stdout() (*progressbar.utils.StreamWrapper* method), 42

wrapped_excepthook (*progressbar.utils.StreamWrapper* attribute), 42

`wrapped_stderr` (*progressbar.utils.StreamWrapper attribute*), [42](#)
`wrapped_stdout` (*progressbar.utils.StreamWrapper attribute*), [42](#)
`wrapper()` (*in module progressbar.widgets*), [53](#)
`WrappingIO` (*class in progressbar.utils*), [42](#)
`writable()` (*progressbar.utils.WrappingIO method*), [43](#)
`write()` (*progressbar.utils.WrappingIO method*), [43](#)
`writelines()` (*progressbar.utils.WrappingIO method*), [43](#)