fastero

Release 0.1.0rc1

Arian Mollik Wasi

Jun 22, 2022

CONTENTS

CHAPTER

ONE

QUICKSTART

Fastero is available on PyPI. To install it to your system:

1. Install with pip

Normal

```
pip install "fastero"
```

This will just install fastero and it's required Dependencies. If you want to export plots, images, or yaml then you need the extra exporting dependencies

With Exporting Dependencies

```
pip install "fastero[export]"
```

This will install matplotlib, pyyaml, selenium, and Pillow alongside fastero. These libraries are required for exporting specific files

2. Install with pipx

Normal

```
pipx install "fastero"
```

This will just install fastero and it's required Dependencies. If you want to export plots, images, or yaml then you need the extra exporting dependencies

With Exporting Dependencies

```
pipx install "fastero[export]"
```

This will install matplotlib, pyyaml, selenium, and Pillow alongside fastero. These libraries are required for exporting specific files

3. Install with pip from github

Normal

```
pip install "git+https://github.com/wasi-master/fastero"
```

This will just install fastero and it's required Dependencies. If you want to export plots, images, or yaml then you need the extra exporting dependencies

With Exporting Dependencies

```
pip install "fastero[export] @ git+https://github.com/wasi-master/fastero"
```

This will install matplotlib, pyyaml, selenium, and Pillow alongside fastero. These libraries are required for exporting specific files

---

Important: Fastero requires python 3.7 and higher

To check if it is installed correctly you can run the following command:

```
fastero --help
```

If that doesn't work, try:

```
python -m fastero --help
```

You may need to replace `python` with your installation specific python command. If that still doesn't work, make sure you have python 3.7+ installed and added to path

CHAPTER

TWO

CLI REFERENCE

This is a hand-written version of the CLI Reference. It may be outdated, if it is please kindly remind me to update it in a github issue or open a pull request. In the case of it being out of date (not having some option or argument), You may want to check out the *Automated CLI Reference*.

This does not cover exporting, those are covered in their own page *Exporting Reference*. They are also covered in *CLI Reference (Automated)*

Also, this version is quite long and filled with examples. :)

2.1 fastero

```
fastero [CODE_SNIPPETS...] [OPTIONS]
```

2.1.1 Arguments

These are the positional arguments, opposed to options, these don't require any prefix and are directly passed

CODE_SNIPPETS

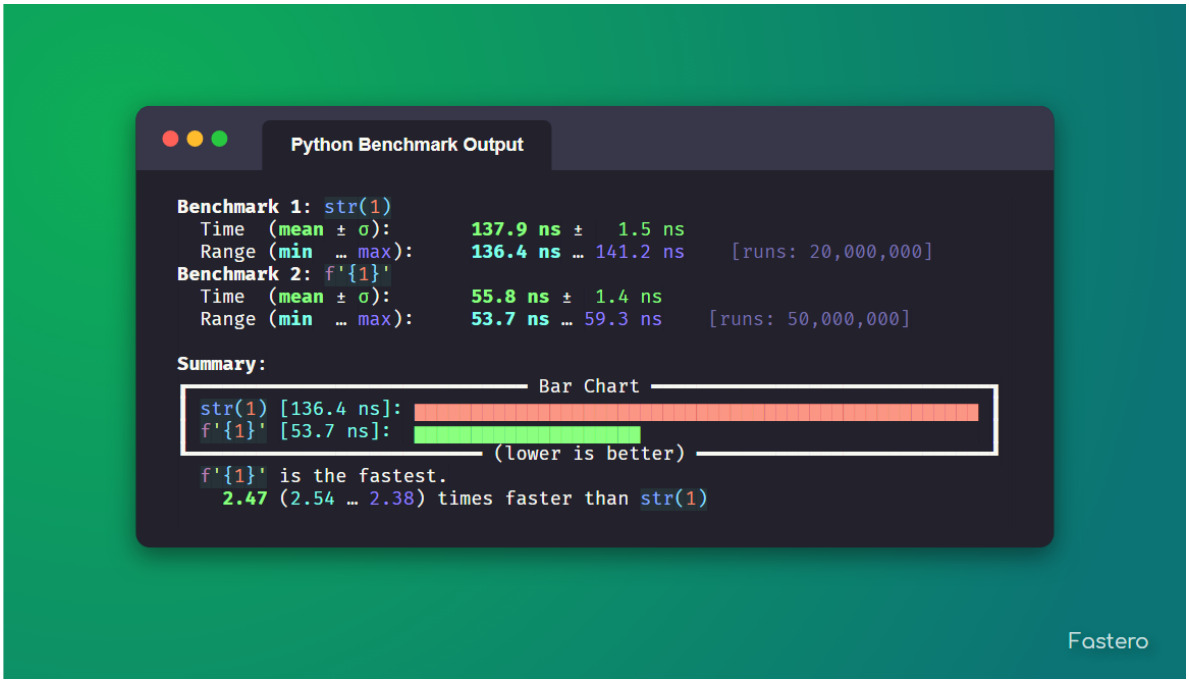
The snippets to benchmark. There may be multiple

Example

```
fastero "str(1)" "f'{1}'"
```

Output

If the image below looks blurry then click it to open it in fullscreen



Any of these can be `-`, to get the input later. Useful for multi-line inputs

---

Example

```
fastero - -
```

Output

If the image below looks blurry then click it to open it in fullscreen

---

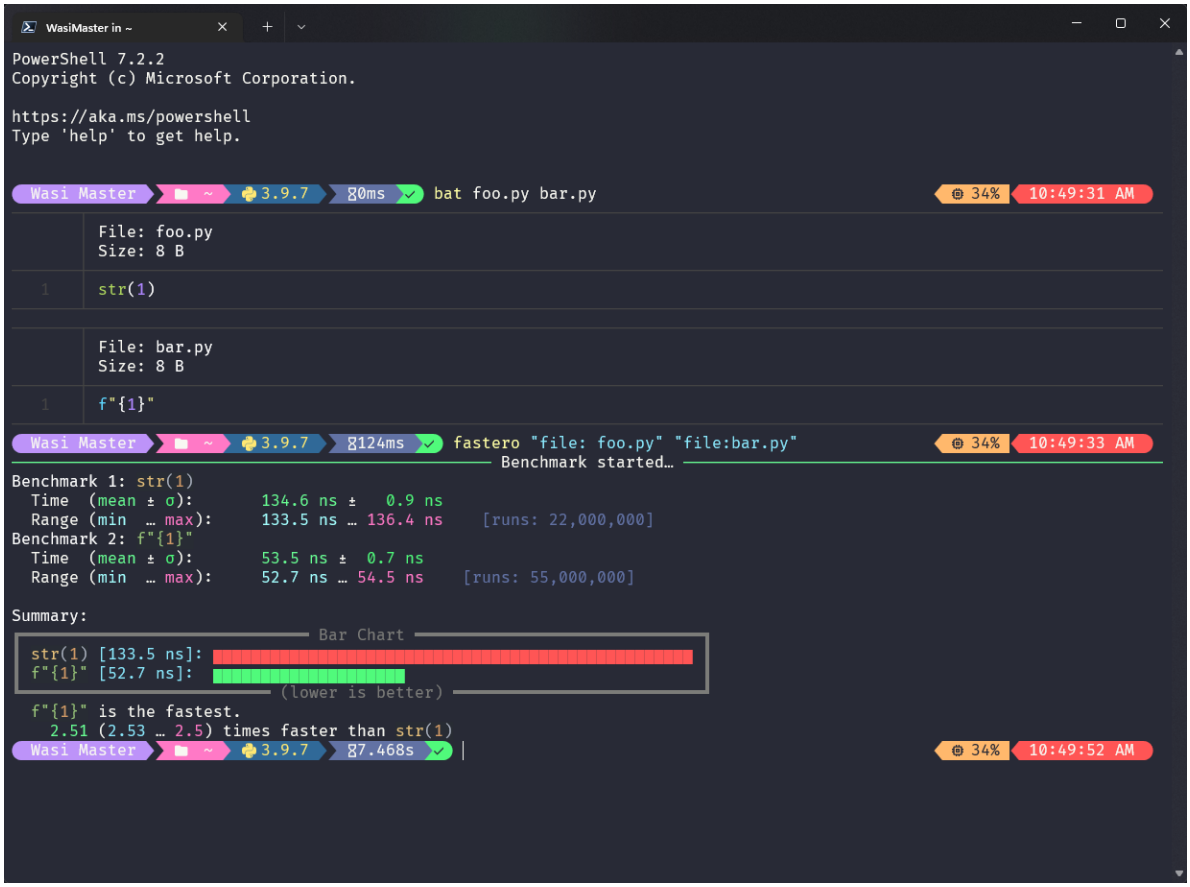
You can also use `file:` to read output from a file. If your code starts with `file:`, you can escape this behavior by adding 2 spaces after the `:`, e.g. `file:  foo`

Example

```
fastero "file: foo.py" "file: bar.py"
```

Output

If the image below looks blurry then click it to open it in fullscreen



The filename for `file:` can also be `stdin` to accept output piped from another program

Example

```
echo "str(1)" | fastero "file: stdin"
```

All of these can also be used together.

Example

```
fastero "str(1)" "file: bar.py" -
```

This would mean the first argument is `str(1)`, the second argument is the contents of bar.py, the third argument is the one given later

2.1.2 Options

-v, --version

Output the version of fastero that is currently being used

-h, --help

Show the help message

-n, --snippet-name <NAME>

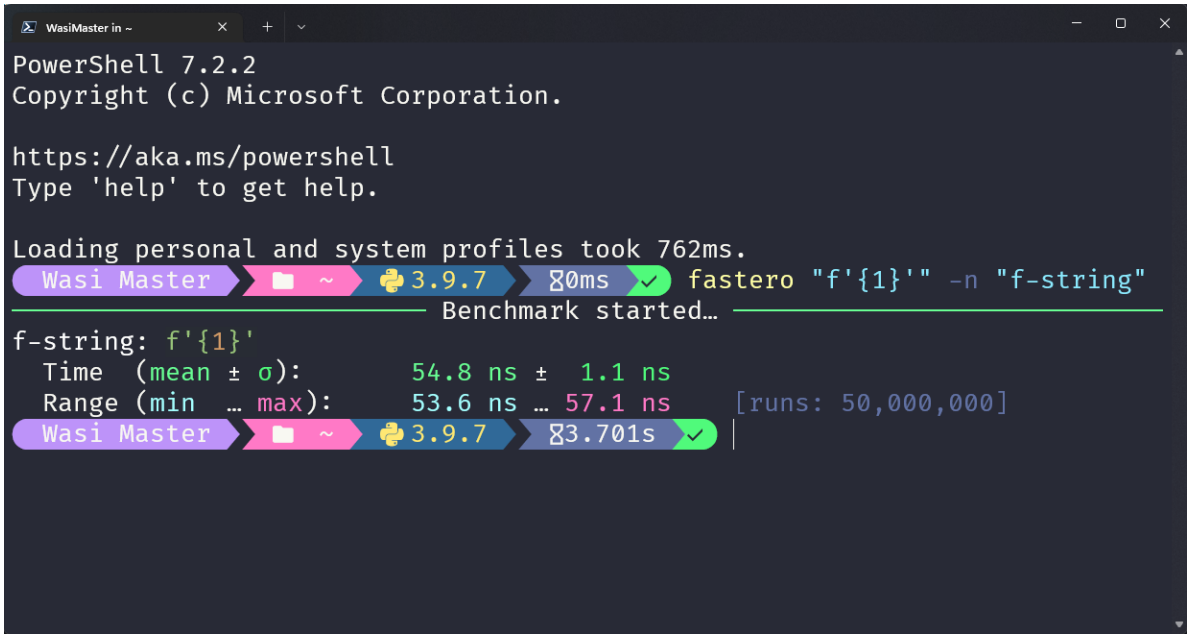
Assign a name to a snippet.

Example

```
fastero "f'{1}'" -n "f-string"
```

Output

If the image below looks blurry then click it to open it in fullscreen



This argument can be used multiple times

Example

```
fastero "f'{1}'" "'{}'.format(1)" -n "f-string" -n "str.format()"
```

Output

If the image below looks blurry then click it to open it in fullscreen

-s, --setup <STMT>

Provide some code to use as the initial setup for the benchmark snippets. This can be used to initialize classes, set variables, import libraries etc.

Default

The default value for setup is `pass`. This is done to be consistent with timeit

---

The format is the exact same as the `CODE_SNIPPETS` argument. Meaning it supports the `file:` directive to get from an file or stdin and the `"-"` parameter to enter multiline input in a prompt

---

Example

```
fastero --setup "l = [0]" "a, = l" "a = l[0]"
```

Output

If the image below looks blurry then click it to open it in fullscreen

-f, --from-json <FILE>

Get input from a json file.

Format

If you only want to get parameters, the format should be this:

```
{
   "setup": "l = [0]",
   "results": [
      {
         "snippet_name": "unpacking",
         "snippet_code": "a, = l"
      },
      {
         "snippet_name": "indexing",
         "snippet_code": "a = l[0]"
      }
   ]
}
```

The keys `snippet_name` and `setup` are optional!

If you however, want to get other information like the *mean* and *standard deviation*, you have to use a json file specifically generated by fastero, or one that uses the same format as the one fastero exports

See also:

`--export-json`, *--json*

Example

Assuming the contents of foo.json are as above:

```
fastero --from-json foo.json
```

Then the output will be the one showed at the end of the `--setup` section

---

You can do a whole bunch of stuff by using this flag. For example if you want to re-preview the results from a json file, you can run

```
fastero --from-json foo.json --only-export
```

Yes I know, this option name is bit unintuitive, since this doesn't have any export parameters, but when I named this option, I thought about what if people want to only export the data from the json file, I am open to renaming suggestions though

If you want to export the results in one of the export formats, then you can add those export options alongside the `--from-json` and `--only-export`, e.g.

```
fastero --from-json foo.json --only-export --export-image
```

So now, it will get the run results from the `foo.json` file and then export a png file with those results

`-j, --json`

Only print json results. This is simillar to the `--export-json` option but instead of exporting to a file, this outputs the json results to standard output. This is given only for scripting purposes. A better reasoning is given in Command Line Interface Guidelines

---

Example

```
fastero "f'{1}'" "'{}'.format(1)" -n "f-string" -n "str.format()" --json
```

Output


```
{
   "setup": "pass",
   "results": [
      {
            "snippet_code": "f'{1}'",
            "snippet_name": "f-string",
            "runs": 55000000,
            "mean": 5.442414363636363e-08,
            "median": 5.392934e-08,
            "min": 5.3144839999999946e-08,
            "max": 6.013294000000001e-08,
            "stddev": 1.926783849689808e-09
      },
      {
            "snippet_code": "'{}'.format(1)",
            "snippet_name": "str.format()",
            "runs": 18000000,
            "mean": 1.652621666666668e-07,
            "median": 1.649461000000003e-07,
            "min": 1.6389370000000002e-07,
            "max": 1.6862390000000005e-07,
            "stddev": 1.424589255715445e-09
```


(continues on next page)

(continued from previous page)

```
      }
   ]
}
```

-q, --quiet

If used, there will be no output printed.

This is useful if you are running it from a script and don't want the output polluting the user's terminal

> **Caution:** Running this without any exporting options or `--json` will just be a waste of time.

-e, --only-export

If used alongside *--from-json*, skips the benchmarking part and just exports the data. The json file needs to to contain the exported data or else this won't work.

-w, --warmup <NUM>

Perform <NUM> warmup runs before the actual benchmark.

Example

```
fastero "f'{1}'" "'{}'.format(1)" -n "f-string" -n "str.format()" --warmup 100_000
```

Tip: The _ is used in replacement of a comma, you can omit it.

Perform this only for presistent improvements. Otherwise all performance gains are lost on each batch. This is due to how fastero benchmarking works. It relies on timeit and timeit doesn't have a warmup parameter, so I'm thinking about subclassing `timeit.Timer` and implementing a warmup parameter myself.

-c, --code-theme <THEME_NAME>

Theme for code input and output, also applicable if "-" is used for any of the parameters,

Default

The default theme is **one-dark**

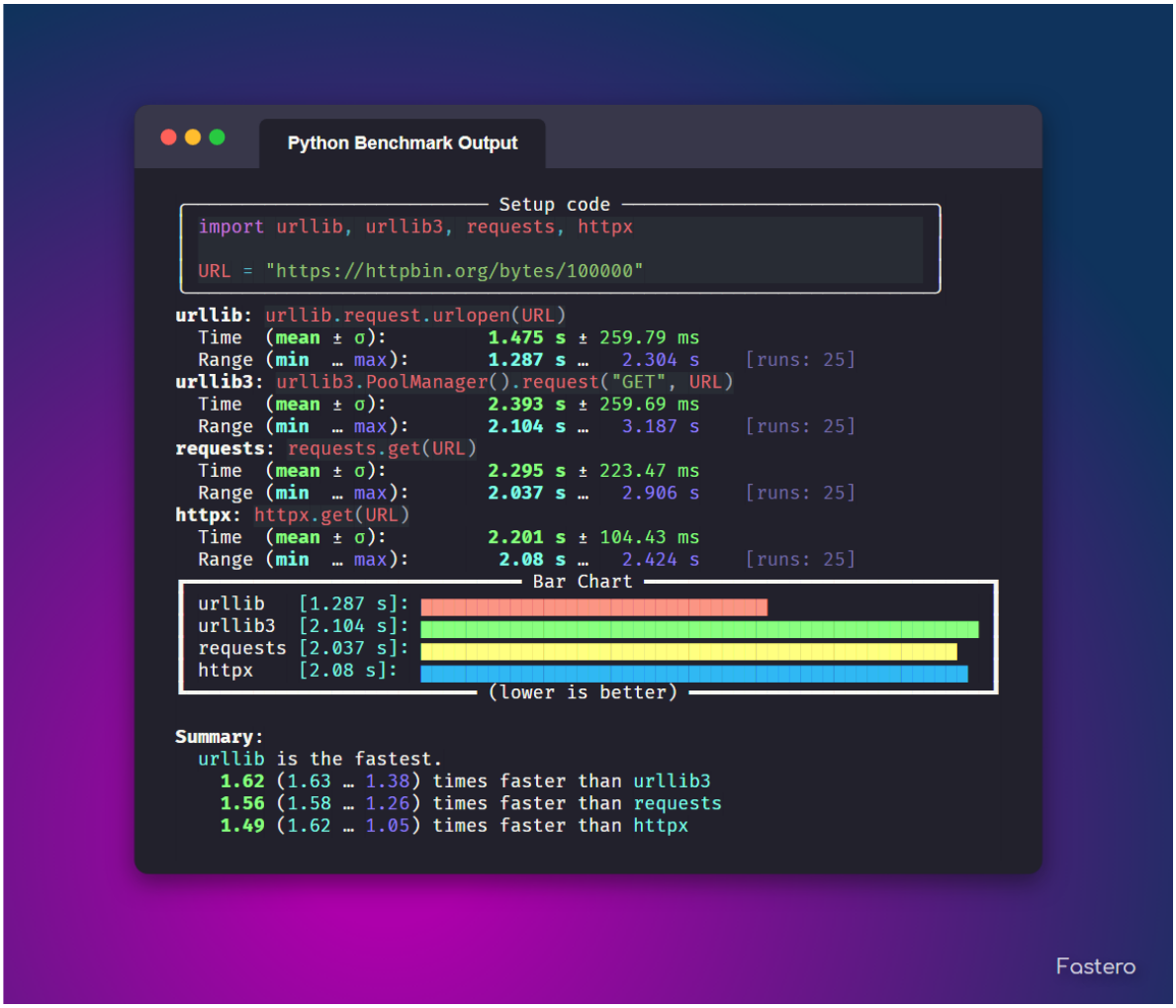
For a list of the themes see https://pygments.org/styles

> **Tip:** Best Themes
>
> These are the best themes: (in my opinion of course)
>
> There are others such as `solarized-dark`, `gruvbox-dark` and many more!

Demonstation

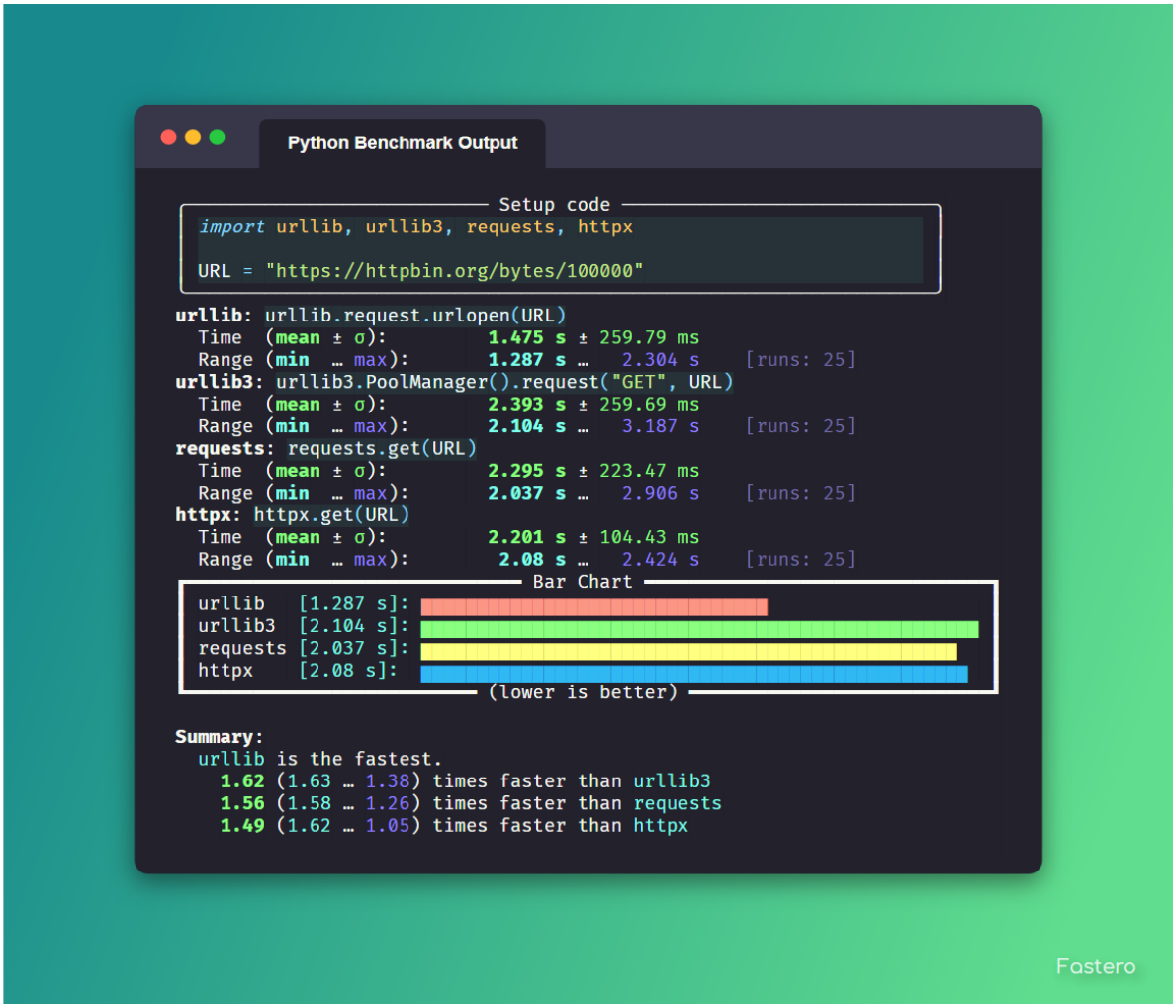
One Dark

This image is basically the console output with this theme, generated by fastero itself



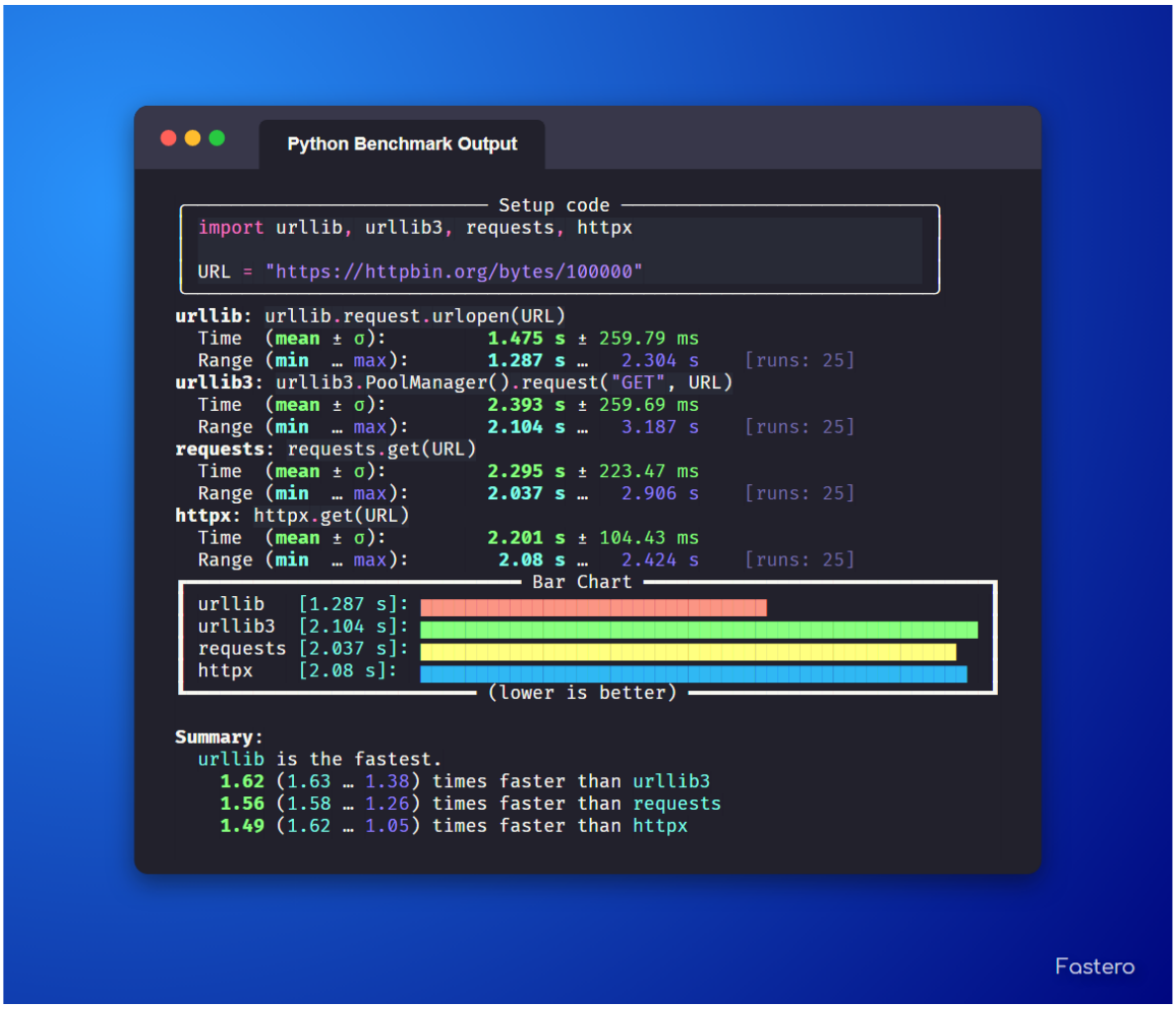
Material

This image is basically the console output with this theme, generated by fastero itself



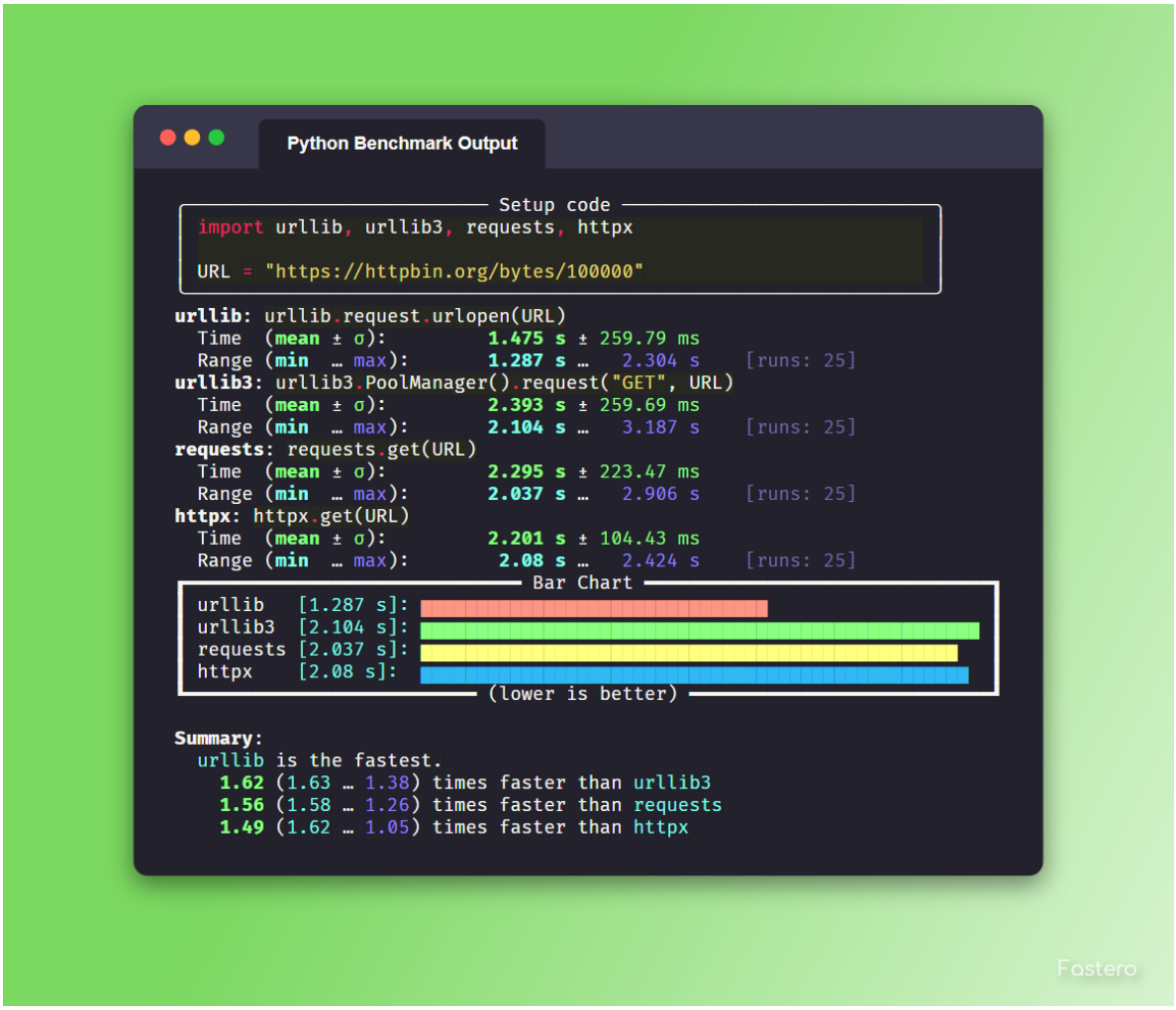
Dracula

This image is basically the console output with this theme, generated by fastero itself



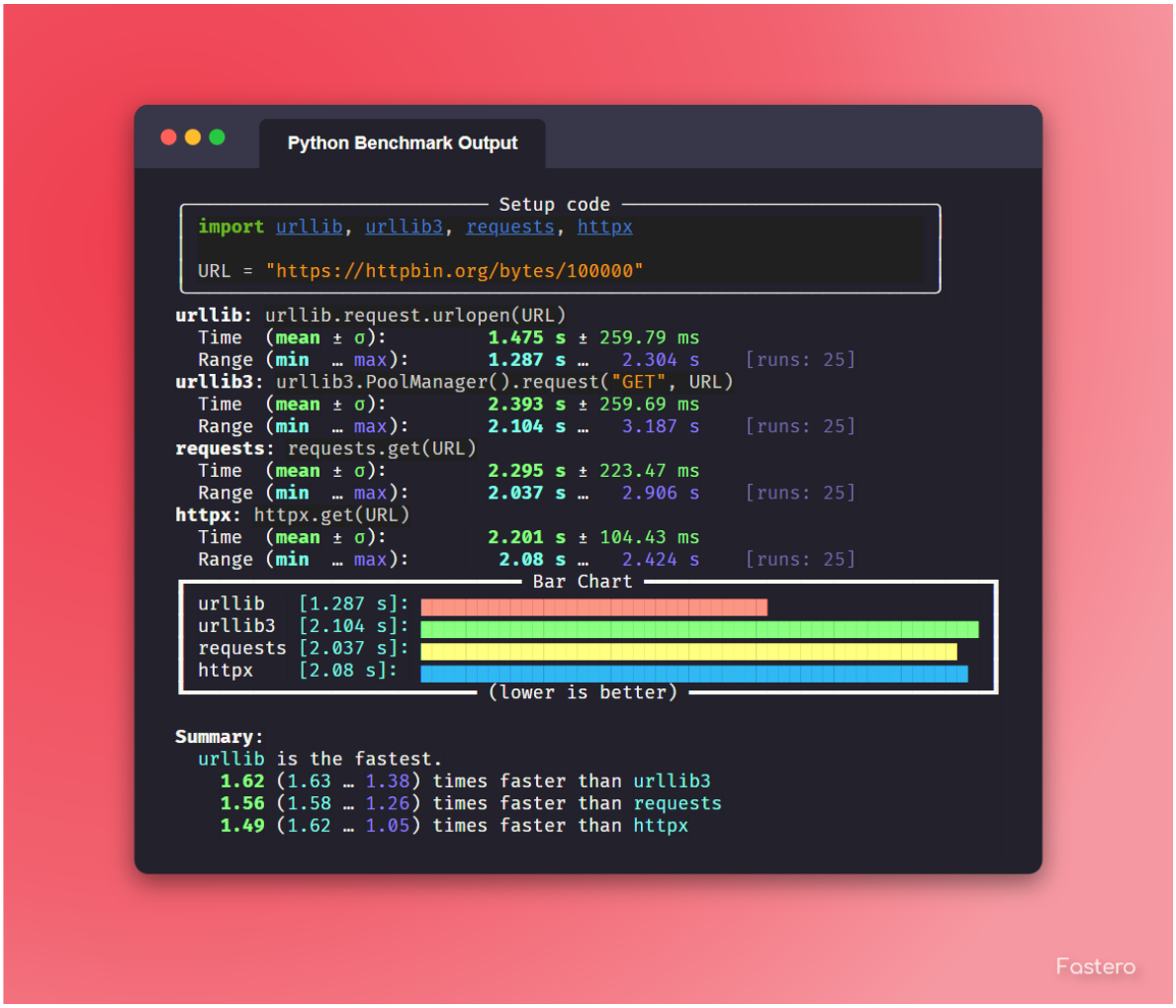
Monokai

This image is basically the console output with this theme, generated by fastero itself



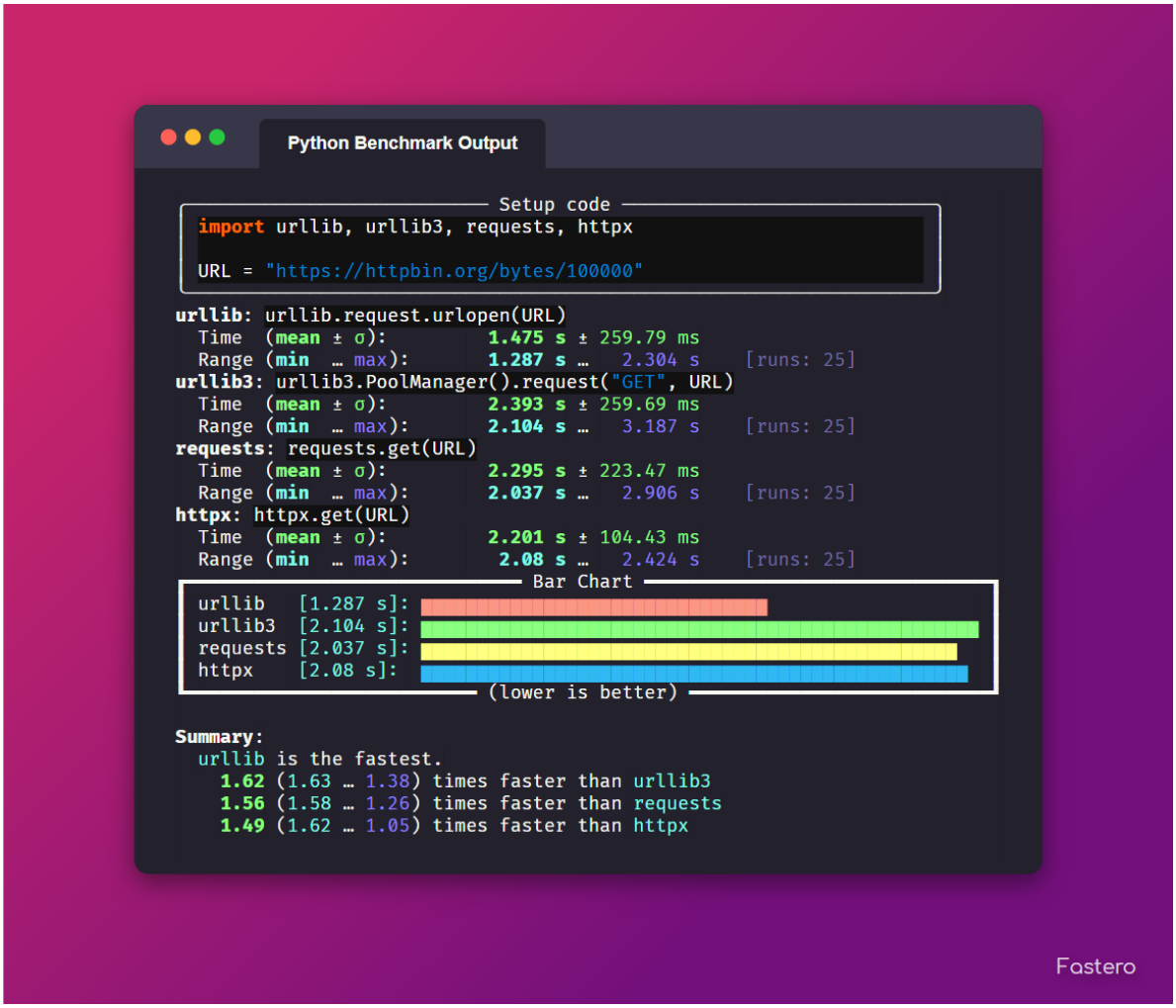
Native

This image is basically the console output with this theme, generated by fastero itself



Fruity

This image is basically the console output with this theme, generated by fastero itself



Example

```
fastero "f'{1}'" "'{}'.format(1)" -n "f-string" -n "str.format()" --code-theme␣
↪monokai
```

`-t, --total-time` <TIME>

How long to test each snippet for. Format: 500ms, 10s, 1m5s, 1.5m, 1h40m15s, etc.

Default

The default duration for benchmarking each code snippet is **3 seconds**

The algorithm is simple, it gets the `--time-per-batch` parameter (by default 200ms), figures out how many runs will be done within that time. Then calculates how many batches will be possible within this <TIME>, and that's your run count, manually specifying *--runs* overrides this. To control the maximum and minimum you can use *--max-runs* and *--min-runs* respectively.

See also:

--time-per-batch

-b, --time-per-batch <TIME>

How long each test batch will last for, increase this to make the tests more accurate at the cost of making progress bar less smooth.

Default

The default duration for each batch is **200 milliseconds**

Also change `--total-time` accordingly or else statistics won't work when it can only do a single batch, therefore it can't determine the mean, median, standard deviation etc.

See also:

--total-time

-u, --time-unit <UNIT>

Set the time unit to be used. Possible values: ns, us, ms, s, dynamic

---

Default

The default the time unit is **dynamic** meaning it depends on the time itself, it is generally the best possible unit for the time.

Applications

This applies to the the console output and the `asciidoc`, `markdown`, `html`, `svg`, `png`, and `plot` export options

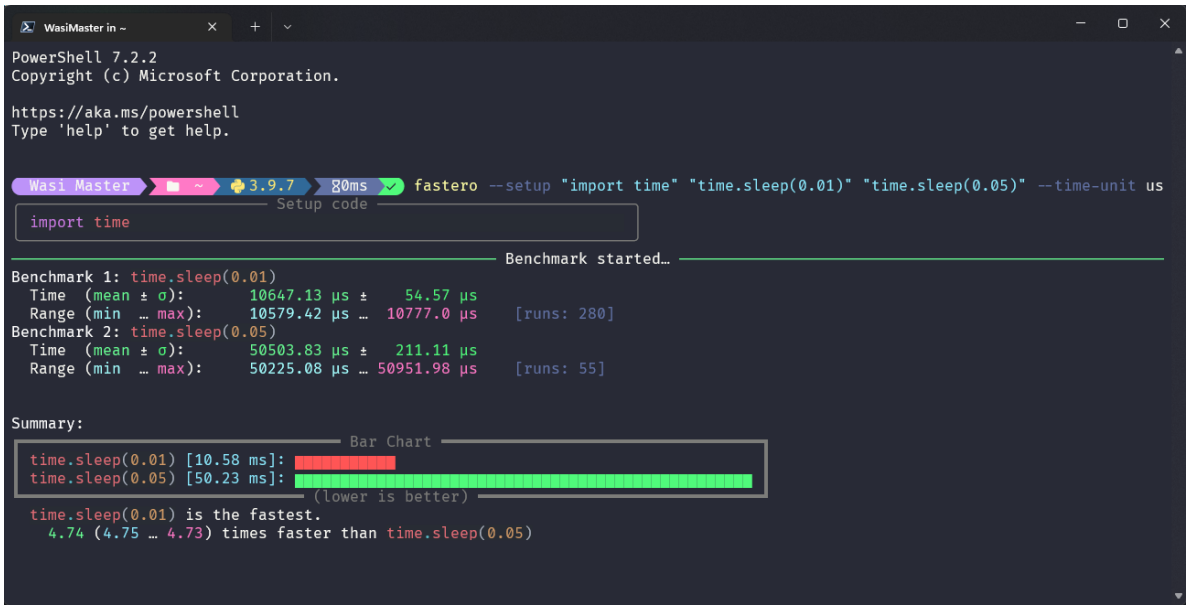
---

Example

```
fastero "f'{1}'" "'{}'.format(1)" -n "f-string" -n "str.format()" --time-unit ms
```

Output

If the image below looks blurry then click it to open it in fullscreen



Available Values

- **ns** (nanoseconds)
- **us** (microseconds)
- **ms** (milliseconds)
- **s** (seconds)
- **dynamic** (dynamic value)

-r, --runs <NUM>

Perform exactly <NUM> runs for each snippet. By default, the number of runs is automatically determined

This is not guaranteed and there may be a maximum of 2 more runs if NUM isn't divisible by 3

-m, --min-runs <NUM>

Perform at least <NUM> runs for each snippet

This exists to aid in controlling the algorithm mentioned in *--total-time*

-m, --min-runs <NUM>

Perform at most <NUM> runs for each snippet

This exists to aid in controlling the algorithm mentioned in *--total-time*

For information about the exporting options, see *Exporting* or if you only want to see the parameters see *CLI Reference (Automated)*

CHAPTER

THREE

CLI REFERENCE (AUTOMATED)

This is a automatically written CLI Reference. It has exactly the same content as the `--help` parameter. There also exists a hand-written CLI Reference at *CLI Reference*. This automated one is given because the hand-written one can be outdated sometimes. The hand-written one is better than this so you should try to use that over this whenever you can

This version does not have many examples. For examples see the hand-written version

3.1 fastero

Benchmark each snippet in **CODE_SNIPPETS**.

Detailed documentation available at https://fastero.readthedocs.io

```
fastero [OPTIONS] [CODE_SNIPPETS]...
```

Options

-n, --snippet-name <NAME>

Give a meaningful name to a snippet. This can be specified multiple times if several snippets are benchmarked.

-s, --setup <STMT>

Code to be executed once in each batch . Execution time of this setup code is *not* timed

Default

pass

-f, --from-json <FILE>

If used, get all the parameters from FILE. The file needs to be a json file with a schema simillar to exported json files

-j, --json

If used, output results in a json format to stdout.

-q, --quiet

If used, there will be no output printed.

-e, --only-export

If used alongside `--from-json`, skips the benchmarking part and just exports the data.

-w, --warmup <NUM>

Perform NUM warmup runs before the actual benchmark. Perform this only for presistent improvements. Otherwise all performance gains are lost on each batch

-c, --code-theme <THEME_NAME>

Theme for code input and output, also applicable if "-" is used for any of the parameters, For a list see https://pygments.org/styles

Default
one-dark

-t, --total-time <TIME>

How long to test each snippet for, specifying `--runs` overrides this. Format: 500ms, 10s, 1m5s, 1.5m, 1h30m15s, etc.

Default
3s

-b, --time-per-batch <TIME>

How long each test batch will last for, increase this to make the tests more accurate at the cost of making progress bar less smooth. Also change `--total-time` accordingly or else statistics won't work

Default
200ms

-u, --time-unit <UNIT>

Set the time unit to be used. Possible values: ns, us, ms, s, dynamic

Default
dynamic

Options
ns | us | ms | s | dynamic

-r, --runs <NUM>

Perform exactly NUM runs for each snippet. By default, the number of runs is automatically determined

-m, --min-runs <NUM>

Perform at least NUM runs for each snippet

Default
2

-M, --max-runs <NUM>

Perform at least NUM runs for each snippet, by default unlimited.

--export-json <FILE>

Export the timing summary statistics as JSON to the given FILE

--export-csv <FILE>

Export the timing summary statistics as CSV to the given FILE.

--export-yaml <FILE>

Export the timing summary statistics as YAML to the given FILE.

--export-markdown <FILE>

Export the timing summary statistics as a Markdown table to the given FILE.

--export-svg <FILE>

Export the console output as a svg image to the given FILE

`--export-image` <FILE>

Export the console output as an image to the given FILE. Exports to svg then uses a headless browser to screenshot that svg output.

`--background` <CSS_COLOR>

Specify a custom background for the generated image. This supports anything the CSS background property supports including images, gradients etc. For more info see https://www.w3schools.com/cssref/css3_pr_background.asp

Default
random

`--selenium-browser` <BROWSER>

The browser to use for exporting the image

Default
chrome

Options
chrome | edge | firefox | opera | safari

`--watermark, --no-watermark`

Whether to add a watermark to the bottom right corner of the generated image. A watermark helps spread the word

Default
True

`--export-asciidoc` <FILE>

Export the timing summary statistics as an AsciiDoc table to the given FILE.

`--export-plot` <FILE>

Export the timing summary statistics as a image of a bar plot to the given FILE

`--label-format` <FORMAT>

Format string for the bar plot, only applicable if the `--export-plot` option is specified.

Default
{snippet_name}\n{snippet_code}

`--dark-background`

If used, the plot background will be in dark mode instead of light

Default
False

`--bar-color` <MATPLOTLIB_COLOR>

A color to use for the bars in the bar plot. Must be in matplotlib supported format, For more info see https://matplotlib.org/stable/tutorials/colors/colors.html

Default
#99bc5a

`--export-html` <FILE>

Export the timing summary statistics as html web page to the given FILE

`-v, --version`

Show the version and exit.

`-h, --help`

Show this message and exit.

Arguments

`CODE_SNIPPETS`
Optional argument(s)

CHAPTER

FOUR

EXPORTING REFERENCE

4.1 Exporting JSON

To export *JSON* to a file you would use the `--export-json` flag

Example

```
fastero "str(1)" "f'{1}'" --export-json foo.json
```

This will save a `foo.json` file with the following contents:

Listing 1: foo.json


```
{
    "$schema": "https://raw.githubusercontent.com/wasi-master/fastero/main/schema.json",
    "setup": "pass",
    "results": [
        {
            "snippet_code": "str(1)",
            "snippet_name": "Benchmark 1",
            "runs": 20000000,
            "mean": 1.3559740499999998e-07,
            "median": 1.3493814999999998e-07,
            "min": 1.3349034999999999e-07,
            "max": 1.400591e-07,
            "stddev": 2.0184569021422298e-09
        },
        {
            "snippet_code": "f'{1}'",
            "snippet_name": "Benchmark 2",
            "runs": 55000000,
            "mean": 5.494544181818183e-08,
            "median": 5.508430000000004e-08,
            "min": 5.336672e-08,
            "max": 5.630708000000002e-08,
            "stddev": 1.0314662950365168e-09
        }
    ]
}
```

This JSON output can then be used to re-run the results using the following command:

```
fastero --from-json foo.json
```

If you wish to not re-run the results but just get the output that was shown previously, run the following command:

```
fastero --from-json foo.json --only-export
```

You can also use other arguments with this!

4.2 Exporting CSV

To export a *CSV* file, use the following command:

Example

```
fastero "str(1)" "f'{1}'" --export-csv foo.csv
```

This will save a `foo.csv` file with the following contents:

Listing 2: foo.csv

```
Snippet Code,Snippet Name,Runs,Mean,Median,Min,Max,Standard Deviation
str(1),Benchmark 1,22000000,1.3751392272727268e-07,1.370651999999999e-07,1.
↪3411479999999997e-07,1.4643009999999999e-07,3.5374505786910588e-09
f'{1}',Benchmark 2,55000000,5.954033636363639e-08,5.4725320000000001e-08,5.
↪307487999999996e-08,8.249068000000008e-08,1.1289950152743191e-08
```

Table 1: CSV Preview

Snippet Code	Snippet Name	Runs	Mean	Median	Min	Max	Standard Deviation
str(1)	Benchmark 1	22000000	1.3751392272727268e-07	1.370651999999999e-07	1.3411479999999997e-07	1.4643009999999999e-07	3.5374505786910588e-09
f'{1}'	Benchmark 2	55000000	5.954033636363639e-08	5.4725320000000001e-08	5.307487999999996e-08	8.249068000000008e-08	1.1289950152743191e-08

4.3 Exporting YAML

To export *YAML* to a file you would use the `--export-yaml` flag

Example

```
fastero "str(1)" "f'{1}'" --export-yaml foo.yaml
```

This will save a `foo.yaml` file with the following contents:

Listing 3: foo.yaml

```yaml
results:
- max: 1.4413549999999997e-07
  mean: 1.4256015499999995e-07
  median: 1.4241862500000002e-07
  min: 1.411376499999999e-07
  runs: 20000000
  snippet_code: str(1)
  snippet_name: Benchmark 1
  stddev: 1.0738769558758217e-09
- max: 8.052079999999985e-08
  mean: 6.093868545454547e-08
  median: 5.8050159999999985e-08
  min: 5.255628000000012e-08
  runs: 55000000
  snippet_code: f'{1}'
  snippet_name: Benchmark 2
  stddev: 9.646527607752279e-09
```

4.4 Exporting Markdown

To export your results as a *Markdown* table, use the `--export-markdown` option

Example

```
fastero 'str(1)' --export-markdown foo.md
```

This will save a `foo.md` file with the following contents:

Listing 4: foo.md

```
|Snippet Code|Snippet Name|Runs|Mean|Median|Min|Max|Standard Deviation|
|---|---|---|---|---|---|---|---|
|str(1)|Benchmark 1|22000000|136.8 ns|135.6 ns|133.7 ns|142.1 ns|2.9 ns|
```

Snippet Code	Snippet Name	Runs	Mean	Median	Min	Max	Standard Deviation
str(1)	Benchmark 1	22000000	136.8 ns	135.6 ns	133.7 ns	142.1 ns	2.9 ns

4.5 Exporting AsciiDoc

To export your results as a *AsciiDoc* table, use the `--export-asciidoc` option

Example

```
fastero "str(1)" --export-asciidoc foo.adoc
```

This will save a `foo.adoc` file with the following contents:

Listing 5: foo.adoc

```
[cols=",,,,,,," options="header"]
|===
|Snippet Code|Snippet Name|Runs|Mean|Median|Min|Max|Standard Deviation
|str(1)|Benchmark 1|20000000|136.5 ns|134.7 ns|134.1 ns|147.7 ns|4.2 ns
|===
```

Snippet Code	Snippet Name	Runs	Mean	Median	Min	Max	Standard Deviation
str(1)	Benchmark 1	20000000	136.5 ns	134.7 ns	134.1 ns	147.7 ns	4.3 ns

4.6 Exporting SVG

To export your console output as a *SVG* file, use the `--export-svg` option

Example

```
fastero "str(1)" "f'{1}'" --export-svg foo.svg
```

This will save a `foo.svg` file with the following contents

Listing 6: foo.svg

```
<svg width="2050.3999999999996" height="670" viewBox="0 0 2050.3999999999996 670"
    xmlns="http://www.w3.org/2000/svg">
    <style>
        @font-face {
            font-family: "Fira Code";
            src: local("FiraCode-Regular"),
                url("https://cdnjs.cloudflare.com/ajax/libs/firacode/6.2.0/woff2/
↪FiraCode-Regular.woff2") format("woff2"),
                url("https://cdnjs.cloudflare.com/ajax/libs/firacode/6.2.0/woff/FiraCode-
↪Regular.woff") format("woff");
            font-style: normal;
            font-weight: 400;
        }
        @font-face {
            font-family: "Fira Code";
            src: local("FiraCode-Bold"),
                url("https://cdnjs.cloudflare.com/ajax/libs/firacode/6.2.0/woff2/
↪FiraCode-Bold.woff2") format("woff2"),
                url("https://cdnjs.cloudflare.com/ajax/libs/firacode/6.2.0/woff/FiraCode-
↪Bold.woff") format("woff");
            font-style: bold;
            font-weight: 700;
        }
        span {
            display: inline-block;
            white-space: pre;
            vertical-align: top;
            font-size: 18px;
            font-family:'Fira Code','Cascadia Code',Monaco,Menlo,'DejaVu Sans Mono',
↪consolas,'Courier New',monospace;
        }
        a {
            text-decoration: none;
            color: inherit;
        }
        .blink {
        animation: blinker 1s infinite;
        }
        @keyframes blinker {
            from { opacity: 1.0; }
            50% { opacity: 0.3; }
            to { opacity: 1.0; }
        }
        #wrapper {
            padding: 140px;
            padding-top: 100px;
        }
        #terminal {
            position: relative;
            display: flex;
```

(continues on next page)

(continued from previous page)

```
            flex-direction: column;
            align-items: center;
            background-color: #0c0c0c;
            border-radius: 14px;
            outline: 1px solid #484848;
        }
        #terminal:after {
            position: absolute;
            width: 100%;
            height: 100%;
            content: '';
            border-radius: 14px;
            background: rgb(71,77,102);
            background: linear-gradient(90deg, #804D69 0%, #4E4B89 100%);
            transform: rotate(-4.5deg);
            z-index: -1;
        }
        #terminal-header {
            position: relative;
            width: 100%;
            background-color: #2e2e2e;
            margin-bottom: 12px;
            font-weight: bold;
            border-radius: 14px 14px 0 0;
            color: #f2f2f2;
            font-size: 18px;
            box-shadow: inset 0px -1px 0px 0px #4e4e4e,
                        inset 0px -4px 8px 0px #1a1a1a;
        }
        #terminal-title-tab {
            display: inline-block;
            margin-top: 14px;
            margin-left: 124px;
            font-family: sans-serif;
            padding: 14px 28px;
            border-radius: 6px 6px 0 0;
            background-color: #0c0c0c;
            box-shadow: inset 0px 1px 0px 0px #4e4e4e,
                        0px -4px 4px 0px #1e1e1e,
                        inset 1px 0px 0px 0px #4e4e4e,
                        inset -1px 0px 0px 0px #4e4e4e;
        }
        #terminal-traffic-lights {
            position: absolute;
            top: 24px;
            left: 20px;
        }
        #terminal-body {
            line-height: 22px;
            padding: 14px;
        }
        .r1 {color: #f2f2f2; text-decoration-color: #f2f2f2;background-color: #0c0c0c;}
```

(continues on next page)

(continued from previous page)

```
.r2 {font-weight: bold;color: #f2f2f2; text-decoration-color: #f2f2f2;;background-color:
↪#0c0c0c;}
.r3 {color: #e5c07b; text-decoration-color: #e5c07b; background-color: #282c34}
.r4 {color: #abb2bf; text-decoration-color: #abb2bf; background-color: #282c34}
.r5 {color: #d19a66; text-decoration-color: #d19a66; background-color: #282c34}
.r6 {color: #0dbc79; text-decoration-color: #0dbc79; font-weight: bold;background-color:
↪#0c0c0c;}
.r7 {color: #0dbc79; text-decoration-color: #0dbc79;background-color: #0c0c0c;}
.r8 {color: #11a8cd; text-decoration-color: #11a8cd; font-weight: bold;background-color:
↪#0c0c0c;}
.r9 {color: #bc3fbc; text-decoration-color: #bc3fbc;background-color: #0c0c0c;}
.r10 {color: #666666; text-decoration-color: #666666;background-color: #0c0c0c;}
.r11 {color: #98c379; text-decoration-color: #98c379; background-color: #282c34}
.r12 {color: #7f7f7f; text-decoration-color: #7f7f7f;color: #f2f2f2; text-decoration-
↪color: #f2f2f2;;background-color: #0c0c0c;}
.r13 {color: #11a8cd; text-decoration-color: #11a8cd; background-color: #0c0c0c}
.r14 {color: #cd3131; text-decoration-color: #cd3131;background-color: #0c0c0c;}
.r15 {color: #11a8cd; text-decoration-color: #11a8cd;background-color: #0c0c0c;}
    </style>
    <foreignObject x="0" y="0" width="100%" height="100%">
        <body xmlns="http://www.w3.org/1999/xhtml">
            <div id="wrapper">
                <div id="terminal">
                    <div id='terminal-header'>
                        <svg id="terminal-traffic-lights" width="90" height="21" viewBox=
↪"0 0 90 21" xmlns="http://www.w3.org/2000/svg">
                            <circle cx="14" cy="8" r="8" fill="#ff6159"/>
                            <circle cx="38" cy="8" r="8" fill="#ffbd2e"/>
                            <circle cx="62" cy="8" r="8" fill="#28c941"/>
                        </svg>
                        <div id="terminal-title-tab">Python Benchmark Output</div>
                    </div>
                    <div id='terminal-body'>
                        <div><span class="r2">Benchmark 1</span><span class="r1">: </
↪span><span class="r3">str</span><span class="r4">(</span><span class="r5">1</span>
↪<span class="r4">)</span><span class="r1">
↪                                                                                </
↪span></div>
<div><span class="r1">  Time  (</span><span class="r6">mean</span><span class="r1"> ± </
↪span><span class="r7"></span><span class="r1">):       </span><span class="r6">138.2 ns
↪</span><span class="r1"> ± </span><span class="r7">  2.2 ns</span><span class="r1">
↪
↪             </span></div>
<div><span class="r1">  Range (</span><span class="r8">min</span><span class="r1">  ...
↪</span><span class="r9">max</span><span class="r1">):     </span><span class="r8">135.
↪6 ns</span><span class="r1"> ... </span><span class="r9">141.6 ns</span><span class="r1
↪">    </span><span class="r10">[runs: 20,000,000]</span><span class="r1">
↪                                                                </span></div>
<div><span class="r2">Benchmark 2</span><span class="r1">: </span><span class="r11">f&
↪#x27;{</span><span class="r5">1</span><span class="r11">}&#x27;</span><span class="r1">
↪
↪                                        </span></div>
```

(continues on next page)

(continued from previous page)

```
<div><span class="r1">  Time  (</span><span class="r6">mean</span><span class="r1"> ± </
↪span><span class="r7"></span><span class="r1">):       </span><span class="r6">54.6 ns
↪</span><span class="r1"> ± </span><span class="r7">  0.8 ns</span><span class="r1">
↪                                                                                 
↪              </span></div>
<div><span class="r1">  Range (</span><span class="r8">min</span><span class="r1">  ...
↪</span><span class="r9">max</span><span class="r1">):     </span><span class="r8">53.9
↪ns</span><span class="r1"> ... </span><span class="r9">55.9 ns</span><span class="r1">
↪    </span><span class="r10">[runs: 50,000,000]</span><span class="r1">
↪                                                                  </span></div>
<div><span class="r1"></span><span class="r1">
↪                                                                                 
↪                   </span></div>
<div><span class="r2">Summary</span><span class="r1">:</span><span class="r1">
↪                                                                                 
↪                                          </span></div>
<div><span class="r12"> Bar Chart </span><span class="r1">
↪                                             </span></div>
<div><span class="r12"></span><span class="r1"> </span><span class="r3">str</span><span
↪class="r4">(</span><span class="r5">1</span><span class="r4">)</span><span class="r1">
↪</span><span class="r13">[135.6 ns]:</span><span class="r1"> </span><span class="r14">
↪</span><span class="r1"> </span><span class="r12"></span><span class="r1">
↪                                                                  </span></div>
<div><span class="r12"></span><span class="r1"> </span><span class="r11">f&#x27;{</span>
↪<span class="r5">1</span><span class="r11">}&#x27;</span><span class="r1"> </span>
↪<span class="r13">[53.9 ns]: </span><span class="r1"> </span><span class="r7">
↪                    </span><span class="r1"> </span><span class="r12"></span><span
↪class="r1">
↪</span></div>
<div><span class="r12">  (lower is better) </span><span class="r1">
↪                                                 </span></div>
<div><span class="r1">  </span><span class="r11">f&#x27;{</span><span class="r5">1</span>
↪<span class="r11">}&#x27;</span><span class="r1"> is the fastest.</span><span class="r1
↪">
↪                                       </span></div>
<div><span class="r1">    </span><span class="r6">2.53</span><span class="r1">  (</span>
↪<span class="r15">2.51</span><span class="r1">  ... </span><span class="r9">2.53</span>
↪<span class="r1">) times faster than </span><span class="r3">str</span><span class="r4
↪">(</span><span class="r5">1</span><span class="r4">)</span><span class="r1">
↪                                                                                 
↪    </span></div>
<div><span class="r1"></span><span class="r1">
↪                                                                                 
↪                   </span></div>
                    </div>
                </div>
            </div>
        </body>
    </foreignObject>
</svg>
```

SVG File Preview:

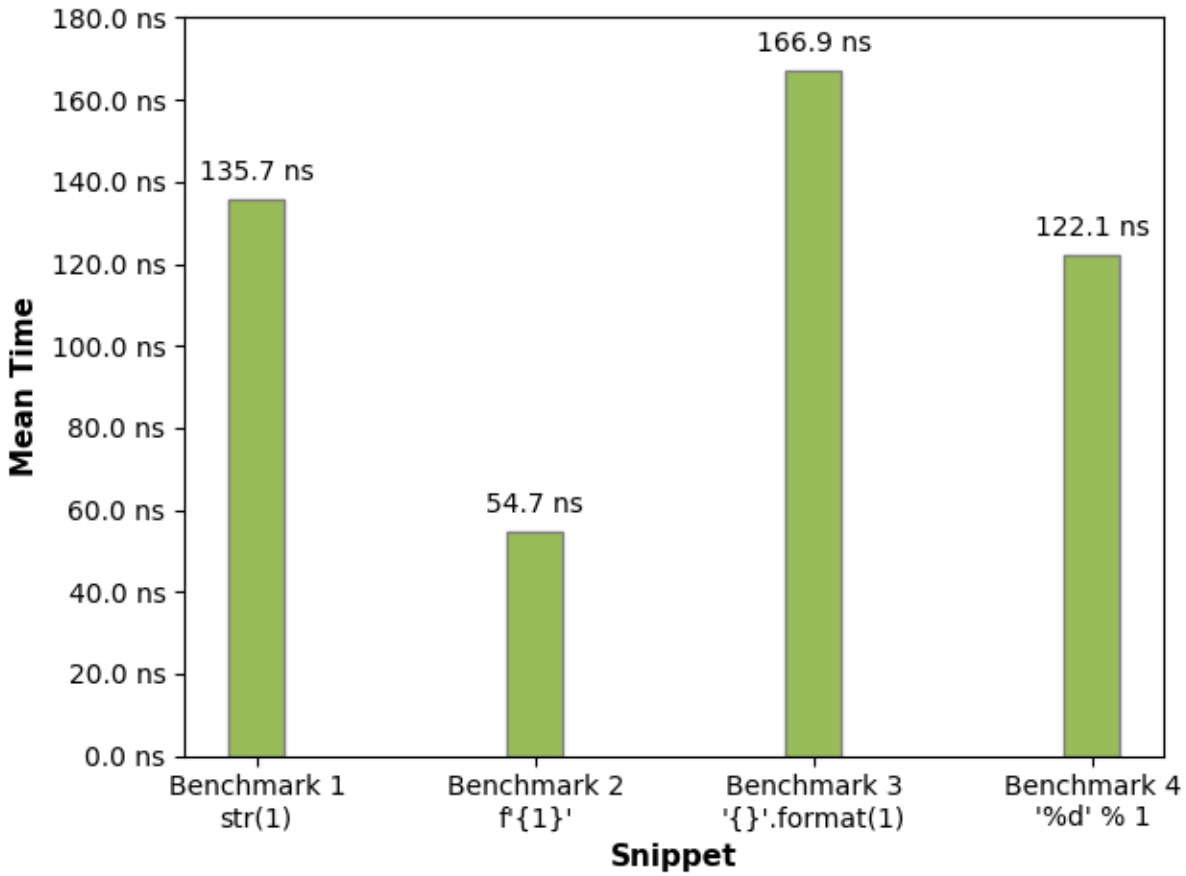
4.7 Exporting a Bar Chart

You can generate a *Bar Chart* using the `--export-plot` command

Example

```
fastero "str(1)" "f'{1}'" "'{}'.format(1)" "'%d' % 1" --export-plot foo.png
```

This will save a `foo.png` file like of the following:

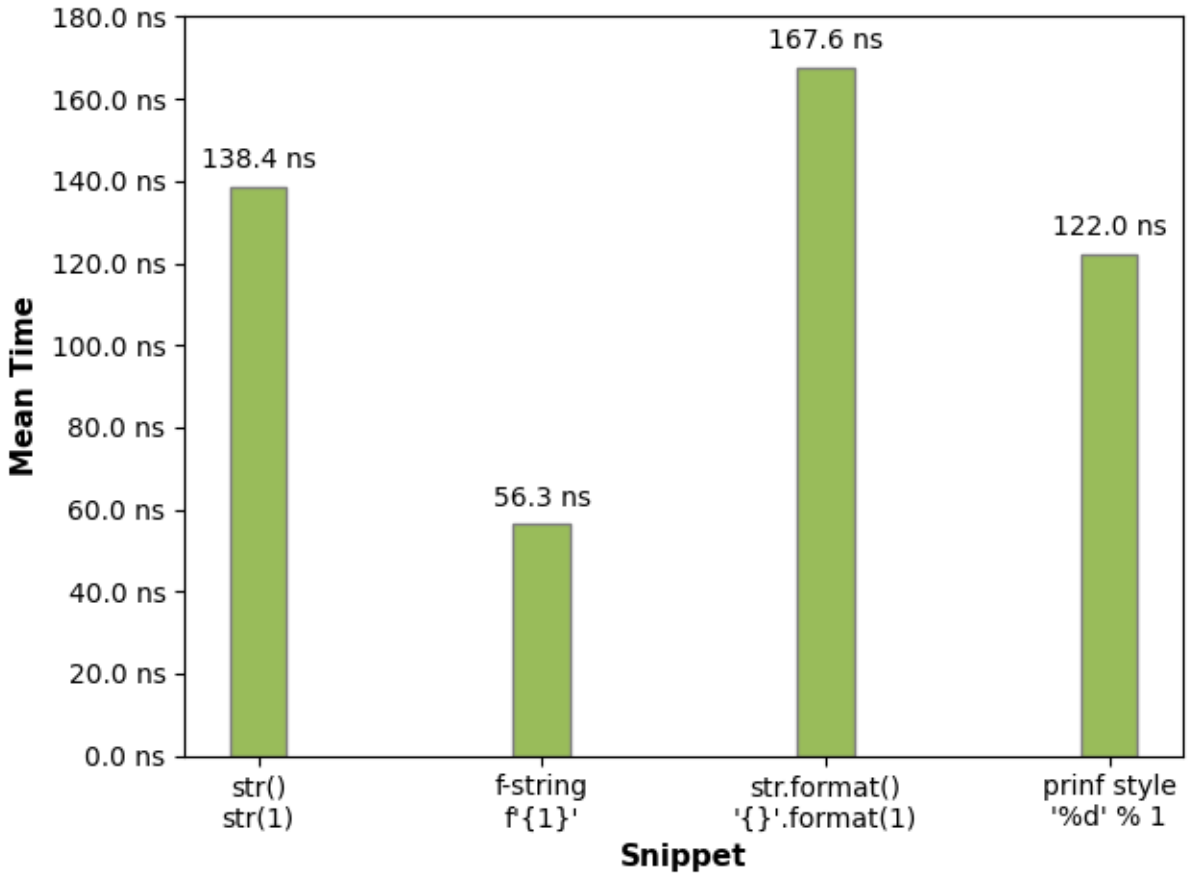


You can (and you should!) add names to your snippets for easier understanding

Example

```
fastero "str(1)" "f'{1}'" "'{}'.format(1)" "'%d' % 1" \
    -n "str()" -n "f-string" -n "str.format()" -n "prinf style" \
    --export-plot foo.png
```

This will save a `foo.png` file like of the following:

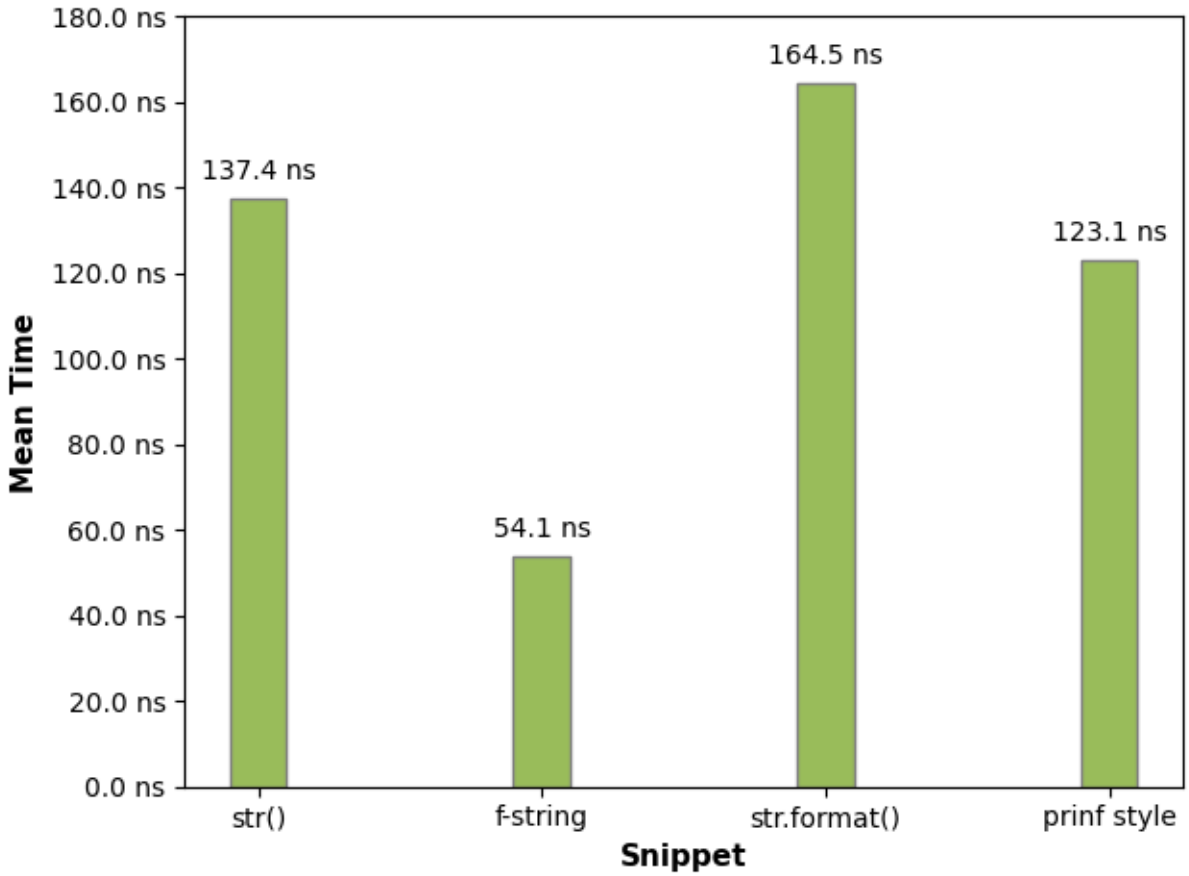


You can also provide a custom label format to use. The default is `{snippet_name}\n{snippet_code}`

Example

```
fastero "str(1)" "f'{1}'" "'{}'.format(1)" "'%d' % 1" \
    -n "str()" -n "f-string" -n "str.format()" -n "prinf style" \
    --export-plot foo.png --label-format "{snippet_name}"
```

This will save a `foo.png` file like of the following:



You can modify the bar color too!

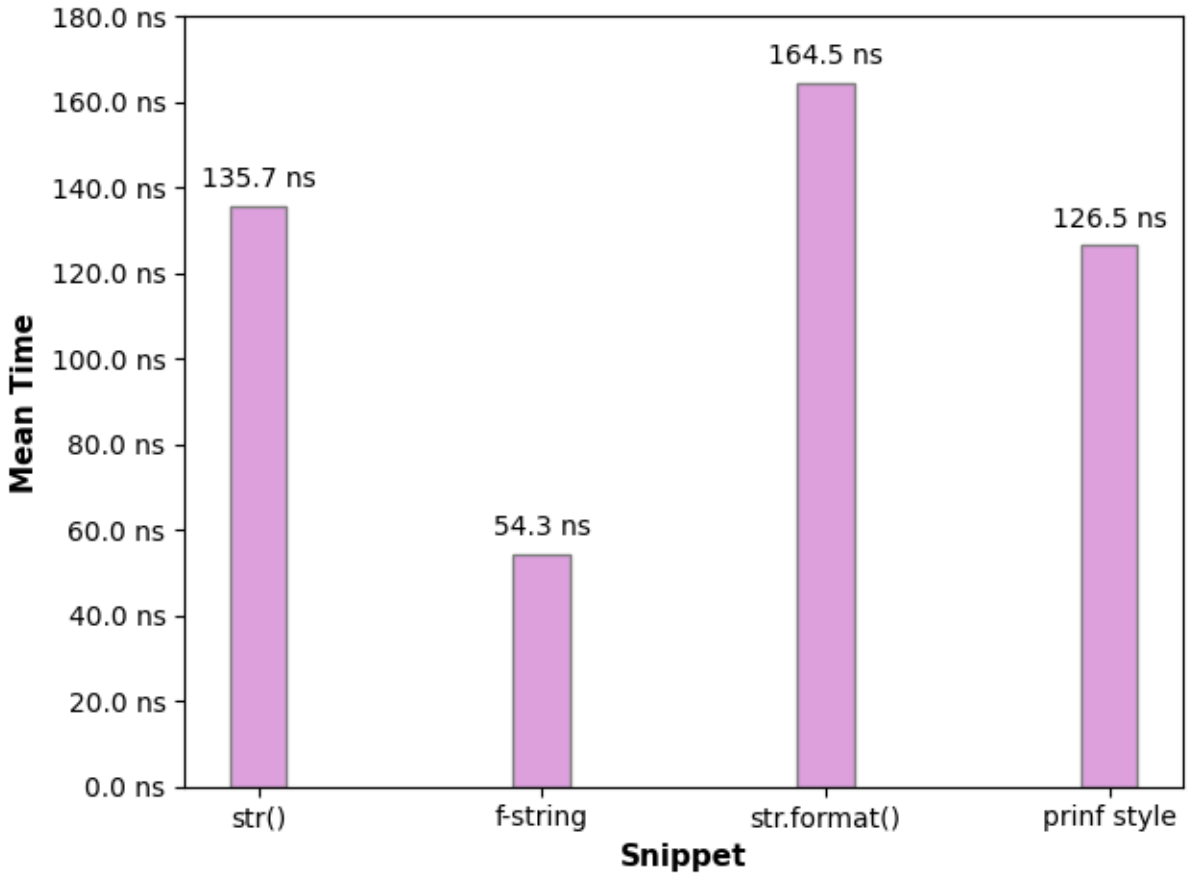
The default color is

For a list of possible color formats and values see matplotlib docs - specifying colors

Example

```
fastero "str(1)" "f'{1}'" "'{}'.format(1)" "'%d' % 1" \
    -n "str()" -n "f-string" -n "str.format()" -n "prinf style" \
    --export-plot foo.png --label-format "{snippet_name}" \
    --bar-color plum
```

This will save a `foo.png` file like of the following:

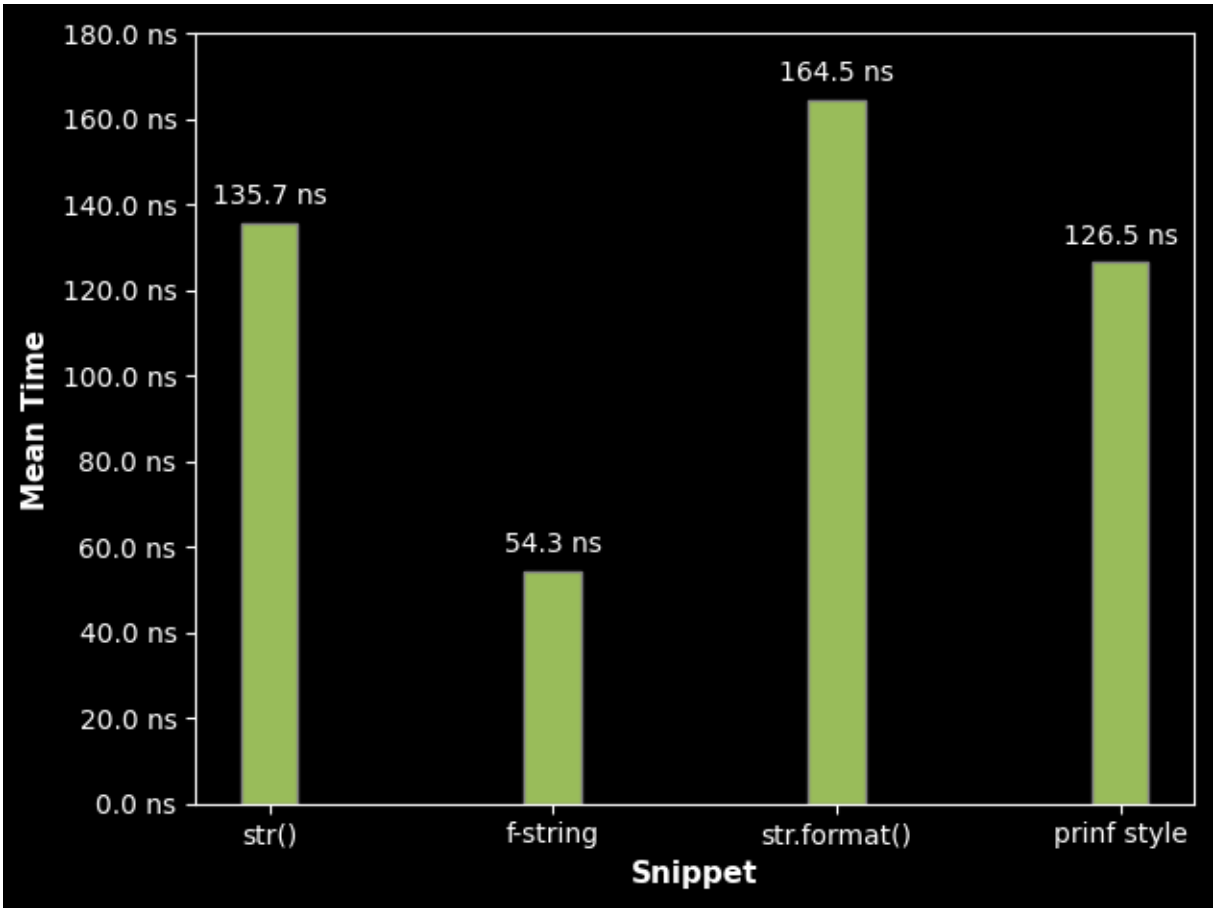


You can change the background color to black using the `dark-background` flag

Example

```
fastero "str(1)" "f'{1}'" "'{}'.format(1)" "'%d' % 1" \
    -n "str()" -n "f-string" -n "str.format()" -n "prinf style" \
    --export-plot foo.png --label-format "{snippet_name}" \
    --dark-background
```

This will save a `foo.png` file like of the following:



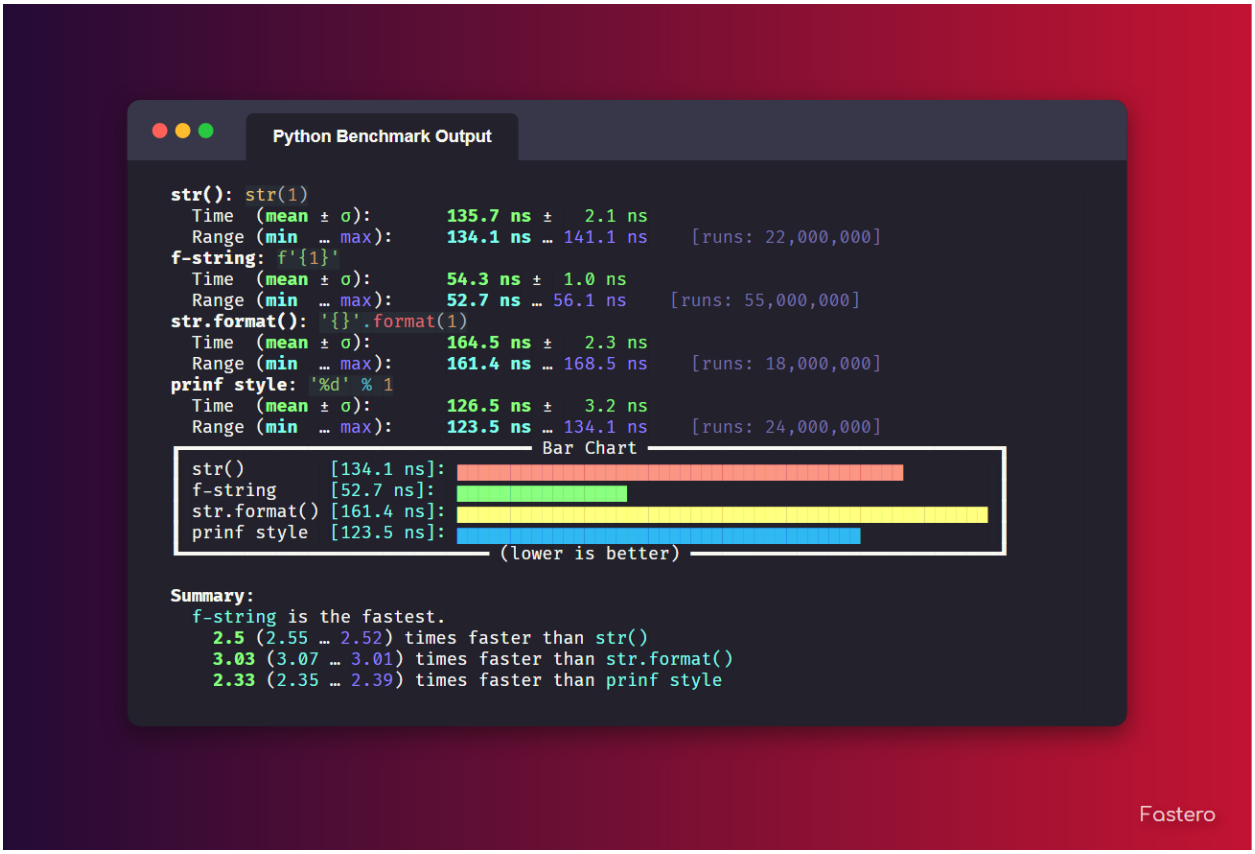
4.8 Exporting an Image

This is in my opinion, the best exporting method! to export an image you should use the `--export-image` flag.

Example

```
fastero "str(1)" "f'{1}'" "'{}'.format(1)" "'%d' % 1" \
    -n "str()" -n "f-string" -n "str.format()" -n "prinf style" \
    --export-image foo.png
```

This will save a `foo.png` file like of the following:



(Open the image in a new tab if it looks blurry)

As you can see there is a watermark for Fastero at the bottom left corner, this can be disabled by using the `--no-watermark` flag.

The way this exporting image works is that it first generates a SVG file using rich, then it opens the SVG in a browser (headless) and takes a screenshot of that browser page. Then it uses PIL to crop out extraneous white borders that the screenshot may have, and then you get the image

Tip: You can resize your terminal window to change the size of the terminal in the image.

You can change which browser it uses using the `--selenium-browser` flag.

Since this uses PIL, the output formats can be anything PIL supports. For a list see Pillow supported formats

You can also specify a custom background using the `--background` flag. This

Example

```
fastero "str(1)" "f'{1}'" "'{}'.format(1)" "'%d' % 1" \
    -n "str()" -n "f-string" -n "str.format()" -n "prinf style" \
    --export-image foo.png --background 'url("https://images.unsplash.com/photo-
↪1649771763042-453b69911ea0")'
```

This will save a `foo.png` file like of the following:

(Open the image in a new tab if it looks blurry)

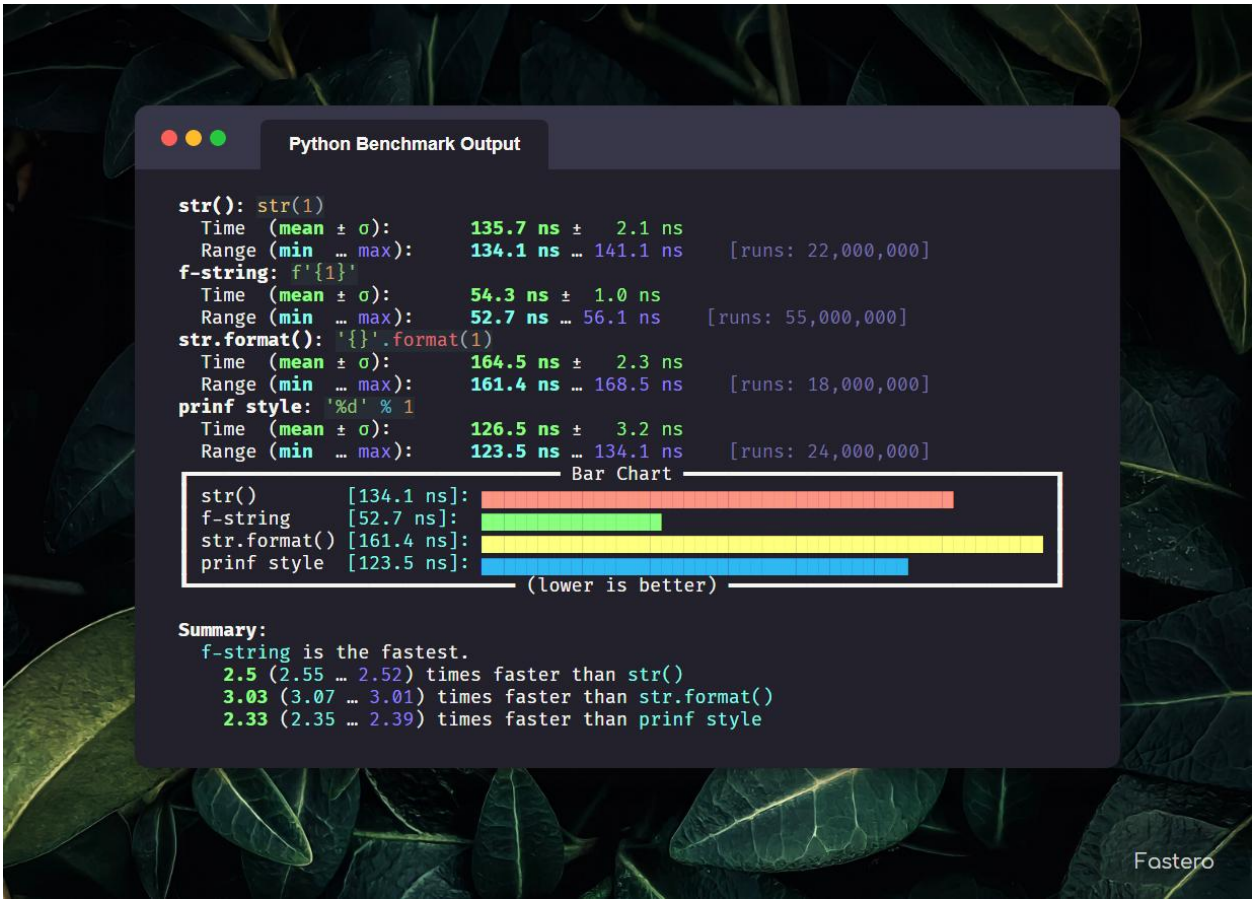


Photo by Eugene Golovesov on Unsplash

CHAPTER

FIVE

TIPS, RECIPIES, AND NOTES

5.1 Supressing output from the benchmark

In your setup, use

```
import sys, os
sys.stdout = sys.stderr = open(os.devnull, 'a', encoding='utf-8')
```

Or in one line:

```
import sys, os; sys.stdout = sys.stderr = open(os.devnull, 'a', encoding='utf-8')
```

5.2 Using stdin as an input

In fastero, "`-`" Is not used for stdin, you should use "`file:  stdin`" instead

5.3 100-400ns overhead

First I'll show you this example

```
Python 3.11.0a6 (main, Mar  7 2022, 16:46:19) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import timeit
>>> timeit.timeit(number=1) * 1_000_000
0.39999940781854093
>>> timeit.timeit(number=1_000_000)
0.010034400002041366
```

Although both of them are supposed to have the same value (in an ideal situation). The first one is almost 40 times lower.

This also applies to `number=2` but the difference is almost cut in half (20 times slower).

```
>>> timeit.timeit(number=2) * 1_000_000
0.39999940781854093
>>> timeit.timeit(number=2_000_000)
0.019674099999974715
```

Due to this reason. Using very low run count will result in a 100-400 nanosecond overhead. This will not matter most of the time. Because, if you are dealing with code so fast that this is gonna matter, the run count will probably be very high so it won't matter anymore. But I would still like to mention it here in case someone's wondering

5.3.1 Programmatic Usage

Although fastero isn't meant to be used programmatically, you can use it as that, with the use of shell commands

```
import os
import json

data = json.loads(os.popen("fastero \"str(1)\" \"f'{1}'\" --json --quiet").read())
print(data['results'][0]['min'])
```

This will output the minimum time required to run "str(1)". The data variable format is same as the format given by the –export-json flg

CHAPTER

SIX

CONTRIBUTING

The code for this library is mostly well commented. If you still have any questions or a problem, DM me on twitter. I also maintain a TODO.md file on the root of the repository and there may be open issues in the github repository so you may want to check those out if you are unsure on what to contribute

6.1 Why contribute?

Open source contribution can be an amazing learning experience. It allows you to give back to and be a part of communities that build valuable open source software. It enables you to make software that you use, better.

6.2 How to contribute?

The most common form of contribution is code. However your contributions do not have to be exclusive to code. You can help by making comments on existing code and bugs, making suggestions by creating an github issue, adding a new gradient to the image export function. Or just tell others about fastero.

If you want a guide on what each file in the codebase does check out *Internal Structure*

6.3 Code of Conduct

Our code of conduct can be viewed at CODE_OF_CONDUCT. If you are not familiar with our Code of Conduct policy, take a minute to read the policy before starting with your first contribution.

CHAPTER

SEVEN

INTERNAL STUCTURE

Fastero's internal structure isn't really the best. It's arguably even worse than my other projects. Most of this project was planned and built as fast as possible without much thought being given to the design and readability of the code.

7.1 Repository Layout

- **`.vscode/`**
  - `settings.json` - Visual Studio Code project specific settings
- **`docs/` - Documentation for the project**
  - **`source/` - Source code for the documentation**
    - `_static/` - Static files for the website
    - `*.rst` - RST files. These are used for the actual content of the documentation
- **`examples/` - Examples on how to use fastero**
  - `export/` - Examples on how the exported data looks
  - `*.ps1` - Powershell script files. These can be ran on Windows
  - `*.sh` - Shell script files. These can be ran on Linux and MacOS
- **`fastero/` - Source code for fastero**
  - `__init__.py` - The `__init__.py` files are required to make Python treat directories containing the file as packages.
  - `__main__.py` - The `__main__.py` is used for python programs that need to be ran from the command line. For example, `python -m fastero`
  - `core.py` - Contains the core code from fastero
  - `exporter.py` - Contains code used for all kinds of different output formats
  - `utils.py` - Short and simple utility functions and classes used by fastero
- `.gitattributes` - File used to tell git to perform LF Normalization
- `.gitignore` - File used to tell git what files to not include in the repository
- `LICENSE` - License information for the source code
- `logo.png` and `logo.jpg` - The logo of fastero
- `README.md` - A guide that gives users a detailed description of the project.
- `schema.json` - The schema used for the JSON export.

- `setup.cfg` - The configuration file for setuptools
- `setup.py` - The python file for setuptools
- `TODO.md` - List of things yet to do

CHAPTER

EIGHT

WORKFLOWS

8.1 Cloning the repository

To clone the repository, make sure you have git installed and available on your system. To clone, run the following command in a terminal

```
git clone "https://github.com/wasi-master/fastero"
```

8.2 Running the library locally

After making some changes to the code or looking at the code, you may want to test out the library locally. To install it from the source code, assuming you are in the root directory of the repository, run the following command:

```
pip install .
```

You may need to change `pip` to `pip3` if you have both python 2 and python 3 installed on your system.

Alternative

Alternatively, you can also use the following command to run the CLI without installing the library first, do note that this only works in the root directory of the repository, to use globally, you have to install it.

```
python -m fastero --help
```

`--help` is just a demo, you can run any command

8.3 Generating the documentation

To build the documentation, you first need to install the requirements from `requirements-docs.txt`

```
python -m pip install -r requirements-docs.txt
```

Then you must change the current working directory to the documentation root directory

```
cd docs
```

Then run the following command to generate the documentation

```
make html
```

This will generate the required HTML files in the `build/html` directory. Go to that directory and open the `index.html` file in an web browser and you should see the documentation

CHAPTER

NINE

LICENSE

9.1 MIT License

CHAPTER

TEN

INDEX

CHAPTER

ELEVEN

GLOSSARY

Fastero
: Fastero is the name of the library. This name was picked after thinking about a lot of things. Namely how easy it is to type, how much finger travel is required, trying to not name it after timeit, keeping it short, typing it with one hand

CLI
: A command-line interface (CLI) is a text-based user interface (UI) used to run programs, manage computer files and interact with the computer.

Argument
: Command line arguments are nothing but simply parameters that are specified after the name of the program in the system's command line

---

Example

```
ls /home/wasi/code
```

Here `ls` is the name of the program and `/home/wasi/code` is the argument

Some programs may accept multiple arguments.

Option
: A command-line option or simply option (also known as a flag or switch) modifies the operation of a command

---

Example

```
ls --color auto
ls --color=auto
```

In both cases `ls` is the name of the program and `--color` is the option, with the value `auto`

Flag
: A command-line flag (sometimes also called a flag). Is basically a option without a value. This also modifies the operation of a command

---

Example

```
ls -a
ls --all
```

In both cases `ls` is the name of the program and `-a` and `--all` are the flags. Note that `-a` is short for `--all`

Run
A single execution of the code snippet. This is simillar to a "loop" in IPython's `%timeit`

Batch
If you're coming from IPython, A batch is like a "run" in IPython's `%timeit` magic function. The way fastero basically works is that it runs `timeit.Timer.timeit()` multiple times which in turn, runs the code `X` number of times, and fastero gets the average from the numbers timeit returns. A batch is basically some `X` amount of *runs* of `timeit.Timer.timeit()` with `X` being a number automatically calculated from the –time-per-batch

Garbage Collection
The process of freeing memory when it is not used anymore. Python performs garbage collection via reference counting and a cyclic garbage collector that is able to detect and break reference cycles. The garbage collector can be controlled using the gc module.

Mean
For a data set, the arithmetic mean, also known as arithmetic average, is a central value of a finite set of numbers: specifically, the sum of the values divided by the number of values.[1]

Standard deviation
In statistics, the standard deviation is a measure of the amount of variation or dispersion of a set of values. A low standard deviation indicates that the values tend to be close to the mean (also called the expected value) of the set, while a high standard deviation indicates that the values are spread out over a wider range.[2]

Standard deviation may be abbreviated SD, and is most commonly represented in mathematical texts and equations by the lower case Greek letter sigma [3]

JSON
JSON stands for JavaScript Object Notation. It is a lightweight data-interchange format. It is easy to parse and generate.

CSV
A CSV (comma-separated values) file is a text file that has a specific format which allows data to be saved in a table structured format.

YAML
YAML stands for "yet another markup language" or "YAML ain't markup language" (a recursive acronym). is a human-friendly, cross language, Unicode based data serialization language designed around the common native data structures of agile programming languages.

Markdown
Markdown is a lightweight markup language that you can use to add formatting elements to plaintext text documents.

AsciiDoc
AsciiDoc is a text document format for writing notes, documentation, articles, books, ebooks, slideshows, web pages, man pages and blogs.

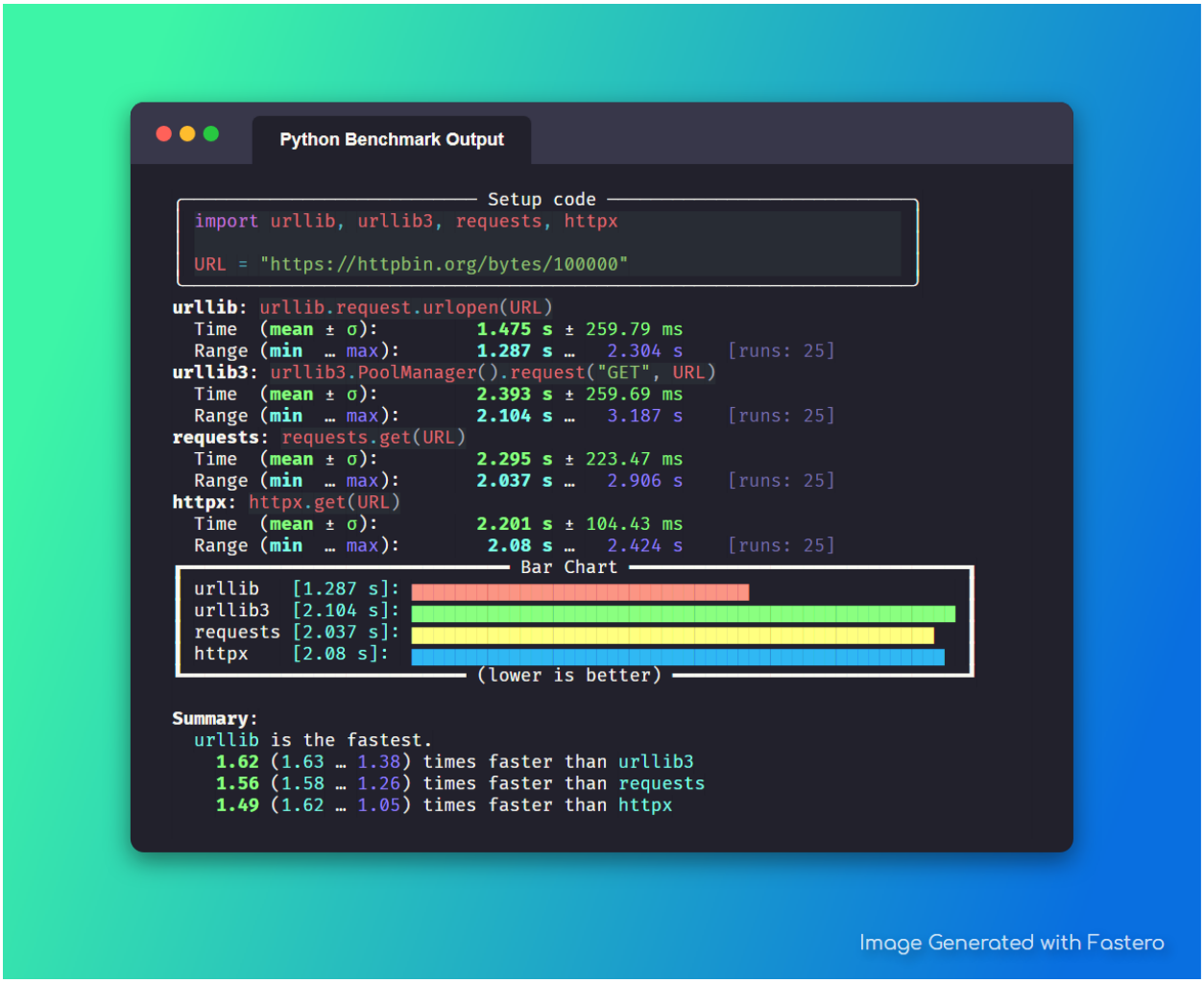
SVG
SVG stands for Scalable Vector Graphics. They are scalable without losing any quality as opposed to raster graphics.

Bar Chart
A bar chart or bar graph is a chart or graph that presents categorical data with rectangular bars with heights or

[1] From https://en.wikipedia.com/wiki/Mean
[2] From https://en.wikipedia.com/wiki/Standard_deviation
[3] From https://en.wikipedia.com/wiki/Standard_deviation

lengths proportional to the values that they represent.

Python timeit CLI for the 21st century.



CHAPTER

TWELVE

INDICES AND TABLES

- **Documentation**
  - Quickstart
  - CLI Reference
  - CLI Reference (Automated)
  - Exporting Reference
  - Tips, Recipies, and Notes
- **Development**
  - Contributing
  - Internal Structure
  - License
  - Workflows
- **Other**
  - Glossary
  - Index

INDEX

Symbols

A

B

C

F

G

J

M

O

R

S

Y