QuickTimer

Release 0.2.0

CribberSix

Jun 24, 2021

CONTENTS:

QuickTimer is an easy to use python package to handle time measurements in code.

CHAPTER

ONE

INTRODUCTION

`QuickTimer` is an easy to use python package to handle time measurements in code.

The aim here was to define a single object which would allow users to easily track passed time throughout their code, as well as to present the tracking results in a fancy way.

1.1 Motivation

As someone who runs a lot of data transformation code and needs to keep an eye on performance during specific steps, I was annoyed by how much time I had to invest again and again to track my code's progress and performance in an orderly fashion.

Existing packages don't offer the ease-of-use I expected from such products or formatted the output horribly, hence this package was born.

CHAPTER

TWO

QUICKTIMER PACKAGE

2.1 Module contents

`class quicktimer.Timer`(*time_unit='timedelta'*, *decimals_percentage=2*, *decimals_time=4*, *output_func=<built-in function print>*)

Bases: `object`

`delete_timestamps`()

Deletes any stored timestamps including descriptions.

`fancy_print`(*delete=True*)

Fancy prints the entire time taken, the differences between the individual timestamps in absolute seconds & in percentages as well as the descriptions.

Parameters **`delete`** (*`bool, optional`*) – deletes the currently stored list of timestamps after ouput, defaults to True

`get_descriptions`()

Returns the stored descriptions.

If no description was supplied when *take_time* was called, the value is an empty String.

Returns List of stored descriptions.

Return type List<str>

`get_timestamps`()

Returns the timestamps including the descriptions as a List of Tuples.

Returns A list of timestamps and discriptions.

Return type List<(datetime, str)>

`set_output_func`(*output_func*)

Sets the output function of the module.

Parameters **`output_func`** (*`function, optional`*) – a function to output messages (e.g. *logger.info* or *print*)

`set_time_unit`(*time_unit*)

Set the unit in which the time is being displayed.

Acceptable values:

- “timedelta”
- “seconds”
- “milliseconds”

Parameters `time_unit` (*str, optional*) – Unit in which time measurements are displayed

Raises `ValueError` – when an unacceptable time_unit is passed as parameter.

`take_time`(*description='', printme=False*)

Snapshots the current time and inserts it into the List as a Tuple with the passed description.

Parameters

- `description` (*str*) – Gets saved alongside the timestamp. Use it as a descriptor of what happened before the function was called, defaults to empty String
- `printme` (*bool*) – Enable printing the description after taking a snapshot of the time. Use this parameter to keep track of the code progress during runtime, defaults to False

CHAPTER

THREE

EXAMPLE

3.1 Installation

The package is available on PyPi :

```
pip install quicktimer
```

3.2 Usage

Instantiate the `Timer` class and insert one-liners with `take_time()` between your existing code to take timestamps.

Call the `fancy_print()` function to print a nicely formatted overview of how much time has passed overall, how much time has passed between the `take_time` calls, including percentage per step and passed step-descriptions.

Although both functions (`take_time()` & `fancy_print()`) can be used without any parameters, you should pass at least a description to `take_time("Finished x!")` to add some context to your measurements.

You can either make use of the default output method (`print` to the console) or you can pass a custom function: for instance to pass the messages to a logger.

3.2.1 Using the default output method

When no `output_func` parameter is passed during instantiation, it defaults to `print` the messages to the console as follows:

```python
import time
from quicktimer import Timer

T = Timer()

# take the starting time
T.take_time(description="The description of the first function-call is not displayed!")

time.sleep(1.1)  # code substitute: parsing the data
T.take_time("Parsed the data")

time.sleep(0.02)  # code substitute
T.take_time()

```

(continues on next page)

(continued from previous page)

```
time.sleep(0.1) # code substitute: Storing the data
T.take_time("Stored the data", True)

T.fancy_print()
```

Output of the code in the console:

```
> Stored the data
> ------ Time measurements ------
> Overall: 0:00:01.254049
> Step 0: 0:00:01.113962 -  88.83 % - Description: Parsed the data
> Step 1: 0:00:00.030001 -   2.39 % - Description:
> Step 2: 0:00:00.110086 -   8.78 % - Description: Stored the data
```

The time can be displayed as `timedelta` (default), `seconds` or `milliseconds`. The number of decimal places for `seconds` or `milliseconds` can be set with the parameter `decimals_time` which defaults to 4. The number of decimal places for the `percentages` can be set with the parameter `decimals_percentage` which defaults to 2.

When initialized with `T = Timer(time_unit="seconds", decimals_time=2, decimals_percentage=1)` the output would be the following.

```
> Stored the data
> ------ Time measurements ------
> Overall: 1.24 seconds
> Step 0: 1.10 seconds -  88.8 % - Description: Parsed the data
> Step 1: 0.03 seconds -   2.5 % - Description:
> Step 2: 0.11 seconds -   8.8 % - Description: Stored the data
```

3.2.2 Using a logger as output method

Instead of `printing` to the console, you can also pass your own function to the module. This can be used with an easily configured `logger` to write the messages to your log.

```
import time
import logging
from quicktimer import Timer

# setting up a logger
my_format = "%(asctime)s [%(levelname)-5.5s]  %(message)s"
logging.basicConfig(filename='test.log', level=logging.INFO, format=my_format)
logger = logging.getLogger()

# logger.info will be used as the output function instead of print
T = Timer(output_func=logger.info)

T.take_time()  # take the starting time
time.sleep(0.5)  # code substitute: parsing the data
T.take_time("Parsed the data")
time.sleep(0.1)  # code substitute: Storing the data
T.take_time("Stored the data", True)

T.fancy_print()
```

The contents of your log-file would look like this:

```
2021-06-24 13:35:43,275 [INFO ]  Stored the data
2021-06-24 13:35:43,275 [INFO ]  ------ Time measurements ------
2021-06-24 13:35:43,275 [INFO ]  Overall: 0:00:00.624691
2021-06-24 13:35:43,275 [INFO ]  Step 0: 0:00:00.512639 -  82.06 % - Description: Parsed␣
↪the data
2021-06-24 13:35:43,275 [INFO ]  Step 1: 0:00:00.112052 -  17.94 % - Description: Stored␣
↪the data
```

CHAPTER FOUR

INDEX

- genindex

PYTHON MODULE INDEX

INDEX