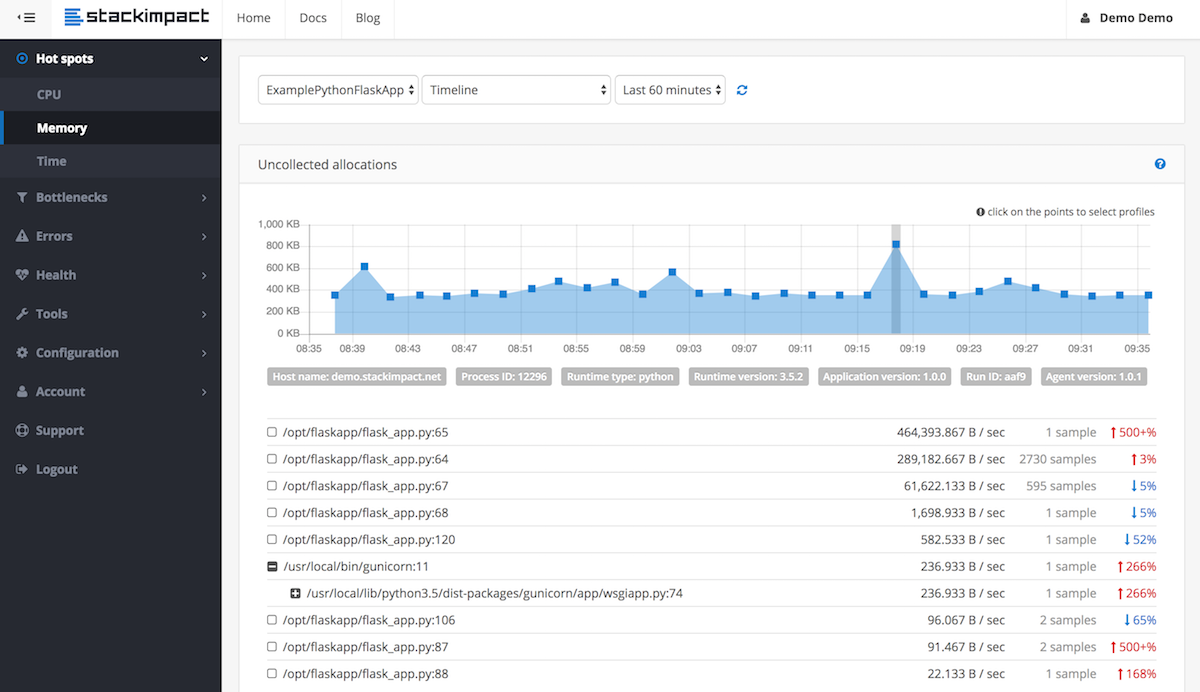
## <https://pyrasite.readthedocs.io/en/latest/>

<https://docs.python.org/3/library/tracemalloc.html#display-the-top-10>

I recommend you use tracemalloc in combination with [pyrasite](https://pyrasite.readthedocs.io/en/latest/). 9 times out of 10, running the [top 10 snippet](https://docs.python.org/3/library/tracemalloc.html#display-the-top-10) in a [pyrasite-shell](https://pyrasite.readthedocs.io/en/latest/Shell.html) will give you enough information and hints to to fix the leak within 10 minutes. Yet, if you're still unable to find the leak cause, pyrasite-shell in combination with the other tools mentioned in this thread will probably give you some more hints too. You should also take a look on all the extra helpers provided by pyrasite (such as the memory viewer).

<https://stackoverflow.com/questions/1435415/python-memory-leaks>

To detect and locate memory leaks for long running processes, e.g. in production environments, you can now use [stackimpact](https://github.com/stackimpact/stackimpact-python). It uses [tracemalloc](https://docs.python.org/3/library/tracemalloc.html) underneath. More info in [this post](https://stackimpact.com/blog/detecting-and-locating-memory-leaks-in-production-python-applications/).



nstall package via pip install pympler

from pympler.tracker import SummaryTracker

tracker = SummaryTracker()

# ... some code you want to investigate ...

tracker.print\_diff()

The output shows you all the objects that have been added, plus the memory they consumed.

Sample output:

types | # objects | total size

====================================== | =========== | ============

list | 1095 | 160.78 KB

str | 1093 | 66.33 KB

int | 120 | 2.81 KB

dict | 3 | 840 B

frame (codename: create\_summary) | 1 | 560 B

frame (codename: print\_diff) | 1 | 480 B

This package provides a number of more features. Check [pympler's documentation](https://pythonhosted.org/Pympler/), in particular the section [Identifying memory leaks](https://pythonhosted.org/Pympler/muppy.html).

<https://stackoverflow.com/questions/552744/how-do-i-profile-memory-usage-in-python>

<https://stackoverflow.com/questions/1435415/python-memory-leaks>

## Tips for effective Python memory management

You can use a memory management library or package to improve the ease with which you can manage your memory usage in Python. This will allow you to monitor the memory usage of individual objects or the amount of memory in the interpreter. For example, memory\_profiler is a package that illustrates the memory usage of your application over time, allowing you to see if you’re using it efficiently. Additionally, implementing generators whenever possible is another effective way to minimize memory demands while working with data sets. You should also manually remove unreferenced objects and any other unnecessary copies of objects.

Understanding the internals of memory management in Python helps in designing memory-efficient applications. It also makes it easier to debug memory issues in an application.

===-----

Pandas chunking

Pandas

### Deleting

Deleting is one of the ways to save space. It might be the best solution, to any memory problem. We may unknowingly save many data frames for training and testing, which is further not used in the process. We can also delete the unused columns. By deleting these can save more space.  We can also delete the null columns present in the data frame which can also lead to saving more space.  We can use the [del keyword](https://www.geeksforgeeks.org/python-del-to-delete-objects/) followed by the item you want to delete. This should remove the item

Pandas transformations and whether instances of previous transformed pd objects reamin in ram due to variables in the code with ref to them BUT at the same time are not need and used anymore in the overall algo

### Stop Loading the whole columns

It is common that we work with larger datasets, but there is no need of loading the entire dataset is necessary. Instead, we can load the specific columns on which you are going to work. By doing this we can restrict the amount of memory consumed to a very low value.

To do this, simply form a temporary dataset that contains only the values that you are going to work on.

Circular references

* **Manual Breakage**: Developers can manually break reference cycles by setting references to None before an object is deleted.

node1.next = None  
node2.next = None  
del node1  
del node2

**Profiling and optimizing memory use**

Understanding and optimizing memory usage are vital for maintaining application performance.

Python offers several tools for memory profiling, which provide insights beyond standard debugging, helping developers identify and resolve memory inefficiencies:

* **pympler**: A comprehensive tool that tracks memory usage and analyzes object space, making it suitable for detailed memory investigations.
* **memory\_profiler**: This tool offers a line-by-line memory usage analysis, allowing developers to pinpoint exact lines where memory consumption is high.
* **tracemalloc**: Integrated into the Python standard library, tracemalloc helps track memory allocations and detect leaks, offering insights into memory allocation trends.

We’ll be using the pympler library in this tutorial, and you can install it using the pip package manager on your terminal:

<https://www.datacamp.com/tutorial/write-memory-efficient-classes-in-python>

<https://www.datacamp.com/tutorial/memory-profiling-python>

**Generators and iterators**

Generators and iterators allow you to create data streams that load one at a time rather than the entire data sequence being stored automatically within the memory. This is advantageous when working with large data sets, helping to maximize the memory you save along the way.

Polars

Dask

pyspark