hashy/cashy

Hashing and Caching

James Abel

SF Python Meetup

April 9, 2025

j@abel.co   @jamesabel   www.abel.co

hashy’s hash

- **Definition: a hash maps data of arbitrary size (like strings, files, or data structures) to a fixed-size string of characters.**
  - Deterministic: The same input always produces the same output.
  - Fixed Size: Regardless of input size, the hash output is always the same size (e.g., 256 bits for SHA-256).
  - Pre-image resistance: Given a hash, it should be practically impossible to reconstruct the original input.
  - Collision resistance: It should be practically impossible to find two different inputs that produce the same hash output.
  - Avalanche effect: A small change in the input will produce a drastically different hash.

```
from hashy import get_dls_sha256

d = {"a": 1, "b": 2, "c": 3}

print(get_dls_sha256(d))

# output:
# e6a3385fb77c287a712e7f406a451727f0625041823ecf23bea7ef39b2e39805
```

Motivation

- Enable caching of computationally intensive tasks
  - Specify input parameters. Create a request and hash it
  - Save the results with a reference to the hash of the input parameters
- Enable caching of large datasets (files)
  - Attach an object’s hash in the cloud. Prior to download, check local cache and use the local cache if a match.
  - This is what AWSimple does. SHA512 for AWS S3 objects (files).

```
request = {"meaning": "life",
           "version": "0.0.1"}

h = get_dls_sha256(request)

answer = run(request)  # will be 42

# hash of “request” is
# aa263b7f5899f006bc6c1b9a926232f324aa3b3935b029a1fc5acb078072215b

# cache “aa263b7f5899f006bc6c1b9a926232f324aa3b3935b029a1fc5acb078072215b.zip”  # 42
```

hashy's hash

- Provides a few wrappers around exiting hash functions.
  - MD5 – short, but can theoretically have collisions and be crackable. Only use if having a short hash is important or you need MD5 compatibility.
  - SHA256 – good tradeoff of hash length and collision resistance/security. Good default selection.
  - SHA512 – may be overkill, but may offer some amount of quantum resistance. Generally, for large files the extra storage over SHA256 is immaterial.
- `hashy` returns a str for consistency and portability.
- Supports str, bytes, file, dict, list, set
  - Supports nested data structures
  - Hashes built-in datatypes that are JSON serializable. Includes numbers.
  - Pre-orders non-ordered data structures (set, dict) for consistency.

hashy’s cachy

- Yet another cache decorator

```
@cachy()
def takes_a_long_time(p: dict[str, int]):
    time.sleep(100)
    return p + 1
```

- Similar to `@functools.cache` but adds:
  - Parameters don’t have to be Python hashable. Works with any hashy hash-able parameters.
  - Local persistent (SQLite, via `SqliteDict`) and in-memory caching.

```
from functools import cache

from hashy import cachy

@cache
def this_will_not_work(data: dict[str: int]) -> int:
    return sum(data.values())

@cachy()
def this_will_work(data: dict[str: int]) -> int:
    return sum(data.values())

d = {"a": 1, "b": 2, "c": 3}

result = this_will_work(d)
print(result)

result = this_will_not_work(d)
print(result)
```

```
6
Traceback (most recent call last):
  File "C:\Users\JamesAbel\projects\hashy\examples\why_hashy_and_caching.py", line 17, in <module>
    result = this_will_not_work(d)
TypeError: unhashable type: 'dict'
```

cachy decorator parameters

```
def cachy(cache_life: Union[timedelta, None] = None,
          cache_dir: Path = get_cache_dir(),
          cache_none: bool = False,
          in_memory: bool = False) -> Callable:
    """
    Decorator to persistently cache the results of a function call, with a cache life.
    :param cache_life: cache life
    :param cache_dir: cache directory
    :param cache_none: cache None results (default is to not cache None results)
    :param in_memory: if True, use an in-memory cache
                      (default is to only use a file-based cache)
    """
```

How to get hashy

pip install hashy

- https://pypi.org/project/hashy/
- https://github.com/jamesabel/hashy
- Tests have 99% code coverage (yay!)

BACKUP