The cache

The cache is one of the reasons why 🤗 Datasets is so efficient. It stores previously downloaded and processed datasets so when you need to use them again, they are reloaded directly from the cache. This avoids having to download a dataset all over again, or reapplying processing functions. Even after you close and start another Python session, 🤗 Datasets will reload your dataset directly from the cache!

Fingerprint

How does the cache keeps track of what transforms are applied to a dataset? Well, 🤗 Datasets assigns a fingerprint to the cache file. A fingerprint keeps track of the current state of a dataset. The initial fingerprint is computed using a hash from the Arrow table, or a hash of the Arrow files if the dataset is on disk. Subsequent fingerprints are computed by combining the fingerprint of the previous state, and a hash of the latest transform applied.

> [!TIP]
> Transforms are any of the processing methods from the How-to Process guides such as Dataset.map() or Dataset.shuffle().

Here are what the actual fingerprints look like:

```
>>> from datasets import Dataset
>>> dataset1 = Dataset.from_dict({"a": [0, 1, 2]})
>>> dataset2 = dataset1.map(lambda x: {"a": x["a"] + 1})
>>> print(dataset1._fingerprint, dataset2._fingerprint)
d19493523d95e2dc 5b86abacd4b42434
```

In order for a transform to be hashable, it needs to be picklable by dill or pickle.

When you use a non-hashable transform, 🤗 Datasets uses a random fingerprint instead and raises a warning. The non-hashable transform is considered different from the previous transforms. As a result, 🤗 Datasets will recompute all the transforms. Make sure your transforms are serializable with pickle or dill to avoid this!

An example of when 🤗 Datasets recomputes everything is when caching is disabled. When

this happens, the cache files are generated every time and they get written to a temporary directory. Once your Python session ends, the cache files in the temporary directory are deleted. A random hash is assigned to these cache files, instead of a fingerprint.

> [!TIP]
> When caching is disabled, use Dataset.save_to_disk() to save your transformed dataset or it will be deleted once the session ends.

Hashing

The fingerprint of a dataset is updated by hashing the function passed to `map` as well as the `map` parameters (`batch_size`, `remove_columns`, etc.).

You can check the hash of any Python object using the fingerprint.Hasher:

```
>>> from datasets.fingerprint import Hasher
>>> my_func = lambda example: {"length": len(example["text"])}
>>> print(Hasher.hash(my_func))
'3d35e2b3e94c81d6'
```

The hash is computed by dumping the object using a `dill` pickler and hashing the dumped bytes.
The pickler recursively dumps all the variables used in your function, so any change you do to an object that is used in your function, will cause the hash to change.

If one of your functions doesn't seem to have the same hash across sessions, it means at least one of its variables contains a Python object that is not deterministic.
When this happens, feel free to hash any object you find suspicious to try to find the object that caused the hash to change.
For example, if you use a list for which the order of its elements is not deterministic across sessions, then the hash won't be the same across sessions either.