Python technical exercise #2 - Asymmetry check

Introduction

Purpose of this exercise is to test candidate's problem-solving and software development skills on a real-world example. Exercise is split into 3 parts, which are meant to build on top of each other, producing a single feature.

Please note: goal of the exercise is to provide a *robust*, *tested* and *easily extensible* implementation of the described feature. In case the allotted time is not enough, please priorize quality over quantity (e.g. two high-quality tasks are better than three low-quality ones).

Guidelines

1. *All function signatures mentioned in the exercise have to be adhered to. Other functions can be introduced, depending on the need.*
2. *Each function mentioned below should be properly documented (docstring) and have a test case that covers it.*
3. *In case of invalid input, lack of data, or any other errors, inform the user by raising an exception.*

Implementation language: Python 3.7+
Duration: 4 hours
Output: ZIP archive with necessary `.py` files, together with a `README` file with instructions on how to execute the code.

Background

Before data can be analysed, its quality needs to be confirmed. One of the necessary conditions is that all reported transactions should be symmetric - amount of goods country **A** reported sending to country **B** should be the same as the amount of goods country **B** reported recieving from country **A**.

Part 1 - Retrieving transaction data

Background

Transaction data can be retrieved from the REST API of the European Central Bank, using the URL below:

`https://sdw-wsrest.ecb.europa.eu/service/data/BSI/Q.HR.N.A.A20.A.1.AT.2000.Z01.E?detail=dataonly`

In the URL above, `Q.HR.N.A.A20.A.1.AT.2000.Z01.E` is a transaction identifier and can be replaced with any other (valid) identifier.

Response, in XML format, contains multiple children of the following format:

```
<generic:Obs>
    <generic:ObsDimension value="2011-Q1"/>
    <generic:ObsValue value="1.219875"/>
</generic:Obs>
```

Each of these represents the transaction amount at a certain point in time.

Task

Implement the function with the following signature:

```
def get_transactions(identifier: str) -> pd.DataFrame
```

Function should fetch the transaction data from the appropriate URL and convert it to a pandas DataFrame, with columns `IDENTIFIER`, `TIME_PERIOD` and `OBS_VALUE`, corresponding to values of the identifier parameter, `generic:ObsDimension` tag and `generic:ObsValue` tag from the XML. `OBS_VALUE` should be converted to `float`.

Example

Running: `get_transactions("Q.DE.N.A.A20.A.1.AT.2000.Z01.E")` should produce the following:

IDENTIFIER	TIME_PERIOD	OBS_VALUE
Q.DE.N.A.A20.A.1.AT.2000.Z01.E	2012-Q2	14001
Q.DE.N.A.A20.A.1.AT.2000.Z01.E	2012-Q3	13568

...

Part 2 - Determine the symmetric identifier

Background

A transaction identifier, such as `Q.HR.N.A.A20.A.1.AT.2000.Z01.E`, consists of several dot-separated components. It is possible to obtain its symmetric counterpart by swapping appropriate components (e.g. source and target, import and export).

Task

Implement the function with the following signature:

```
def get_symmetric_identifier(
    identifier: str,
    swap_components: dict[int, int]
) -> str
```

The function should return a new identifier, obtained from the provided one, by swapping components, as indicated by the key-value pairs in the provided dictionary. The pairs represent 0-based indices of components to swap.

Example

Running:

`get_symmetric_identifier("Q.HR.N.A.A20.A.1.AT.2000.Z01.E", {1: 7, 2: 3})`

should produce the following: `"Q.AT.A.N.A20.A.1.HR.2000.Z01.E"`

Part 3 - Finding asymmetries

Task

Implement the function with the following signature:

```python
def get_asymmetries(
    identifier: str,
    swap_components: dict[int, int]
) -> pd.DataFrame
```

The function should determine the symmetric counterpart of the provided identifier using the function from part #2 and fetch data for both, using the function from part #1.

After fetching the data, construct the DataFrame with the following columns:

- **TIME_PERIOD**: Index
- **PROVIDED_ID**: Provided identifier
- **SYMMETRIC_ID**: Symmetric identifier
- **DELTA**: Absolute difference between `OBS_VALUE` columns of the provided and symmetric identifier, for the same `TIME_PERIOD`

In case a `TIME_PERIOD` value exists in one DataFrame, but not in the other, it should be skipped.

Example

Running:

```
get_asymmetries("Q.HR.N.A.A20.A.1.AT.2000.Z01.E", {1: 7})
```

should produce the following:

TIME_PERIOD	PROVIDED_ID	SYMMETRIC_ID	DELTA
2014-Q1	Q.HR.N.A.A20.A.1.AT.2000.Z01.E	Q.AT.N.A.A20.A.1.HR.2000.Z01.E	9036,292580
2014-Q2	Q.HR.N.A.A20.A.1.AT.2000.Z01.E	Q.AT.N.A.A20.A.1.HR.2000.Z01.E	8809,512144

...

Note: `TIME_PERIOD` *2013-Q4* exists in `Q.HR.N.A.A20.A.1.AT.2000.Z01.E`, but not in `Q.AT.N.A.A20.A.1.HR.2000.Z01.E` so it has been omitted.