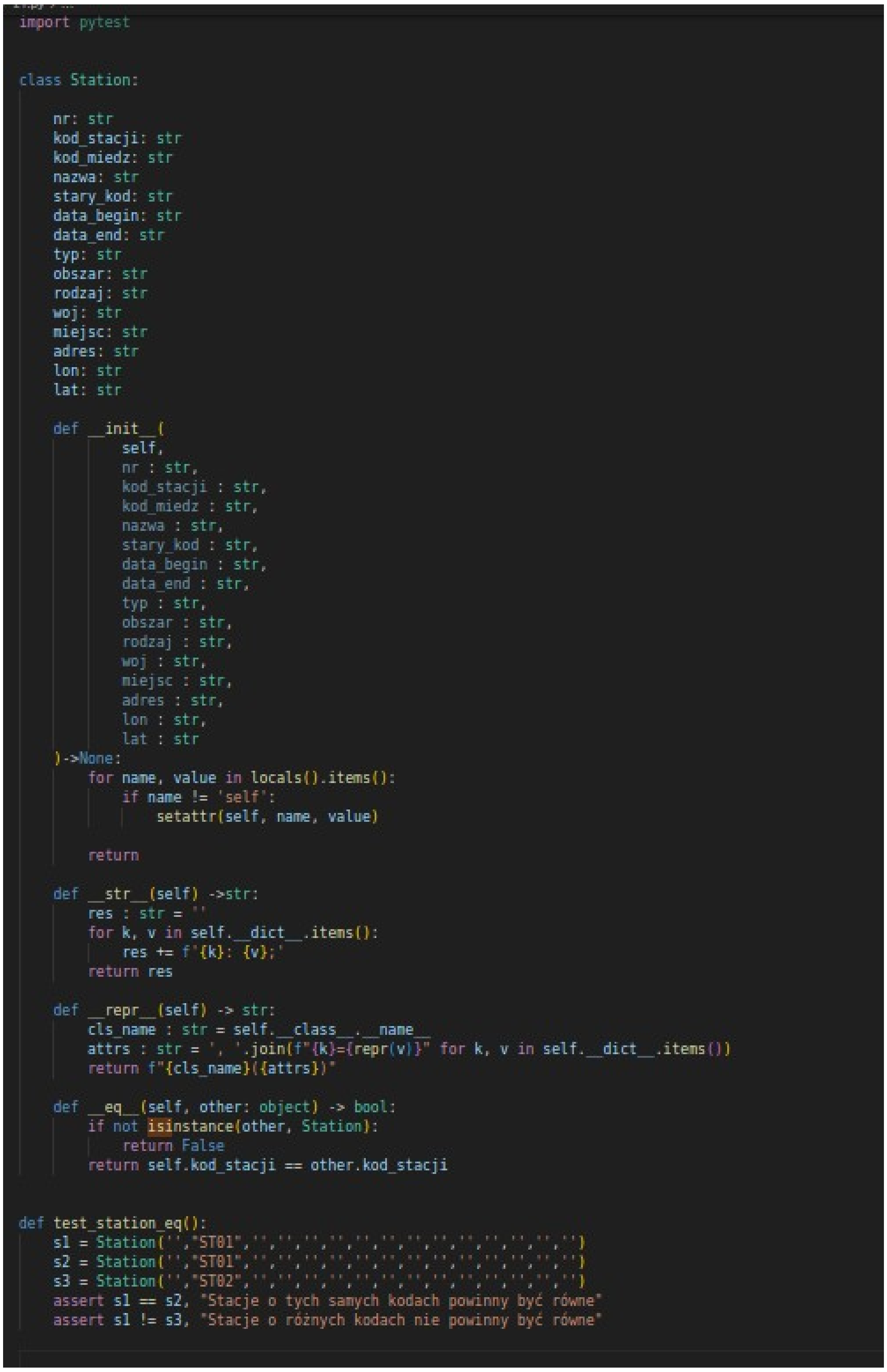
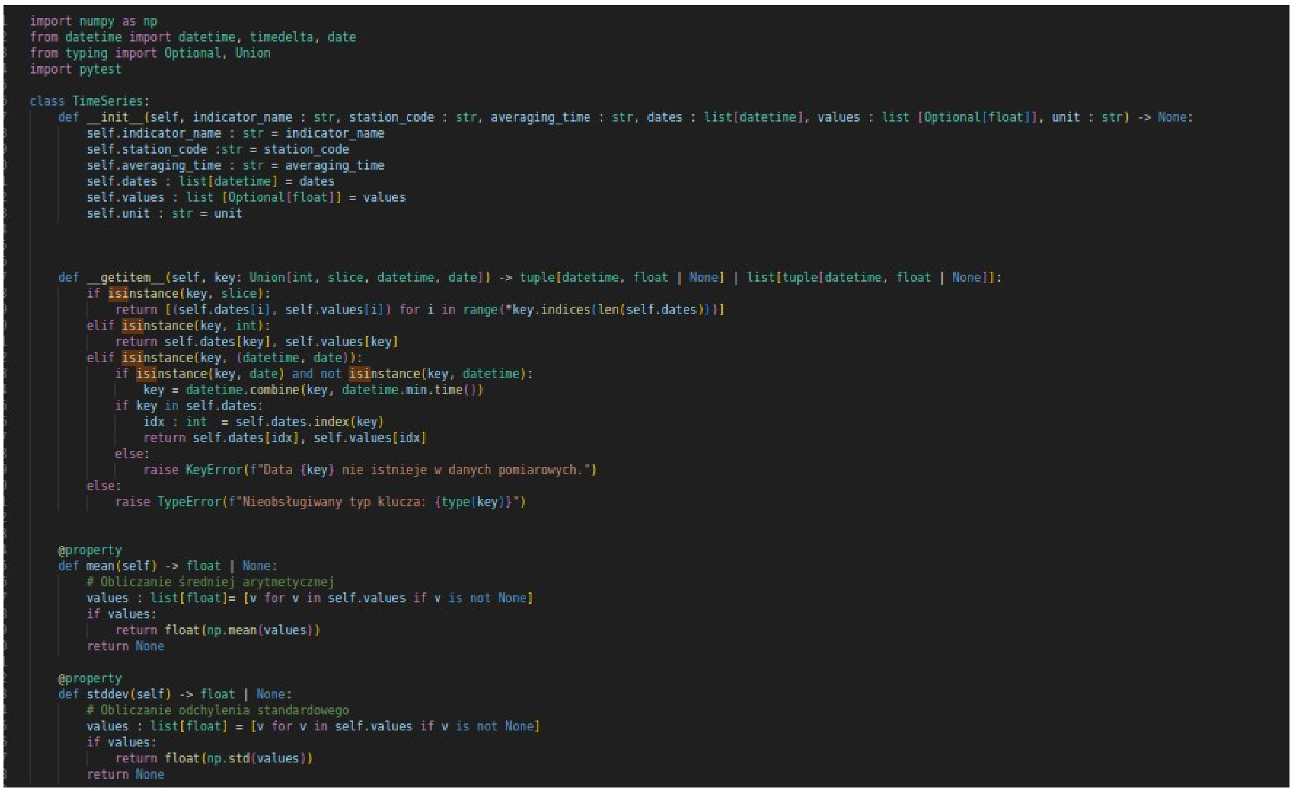
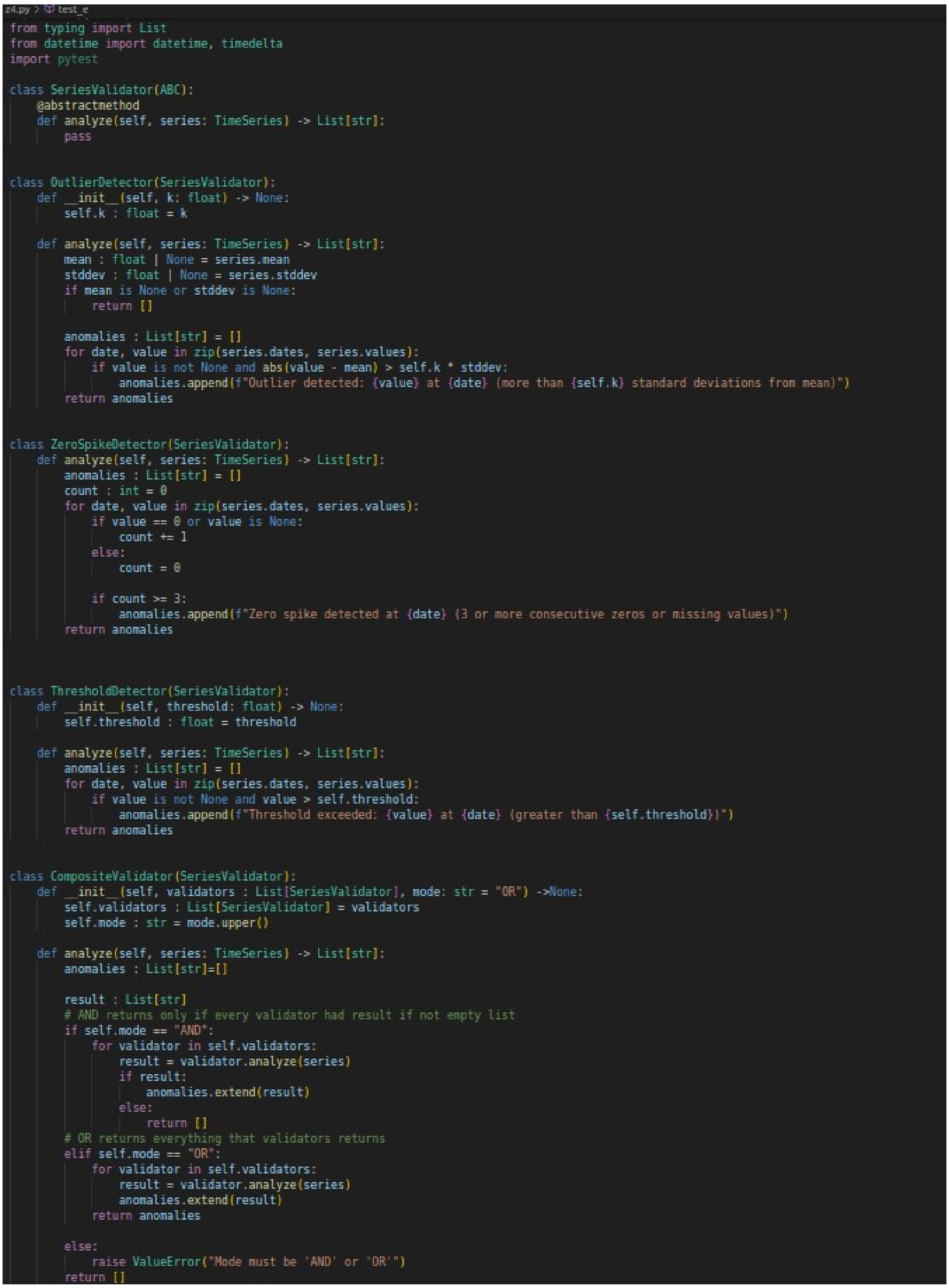
Z1



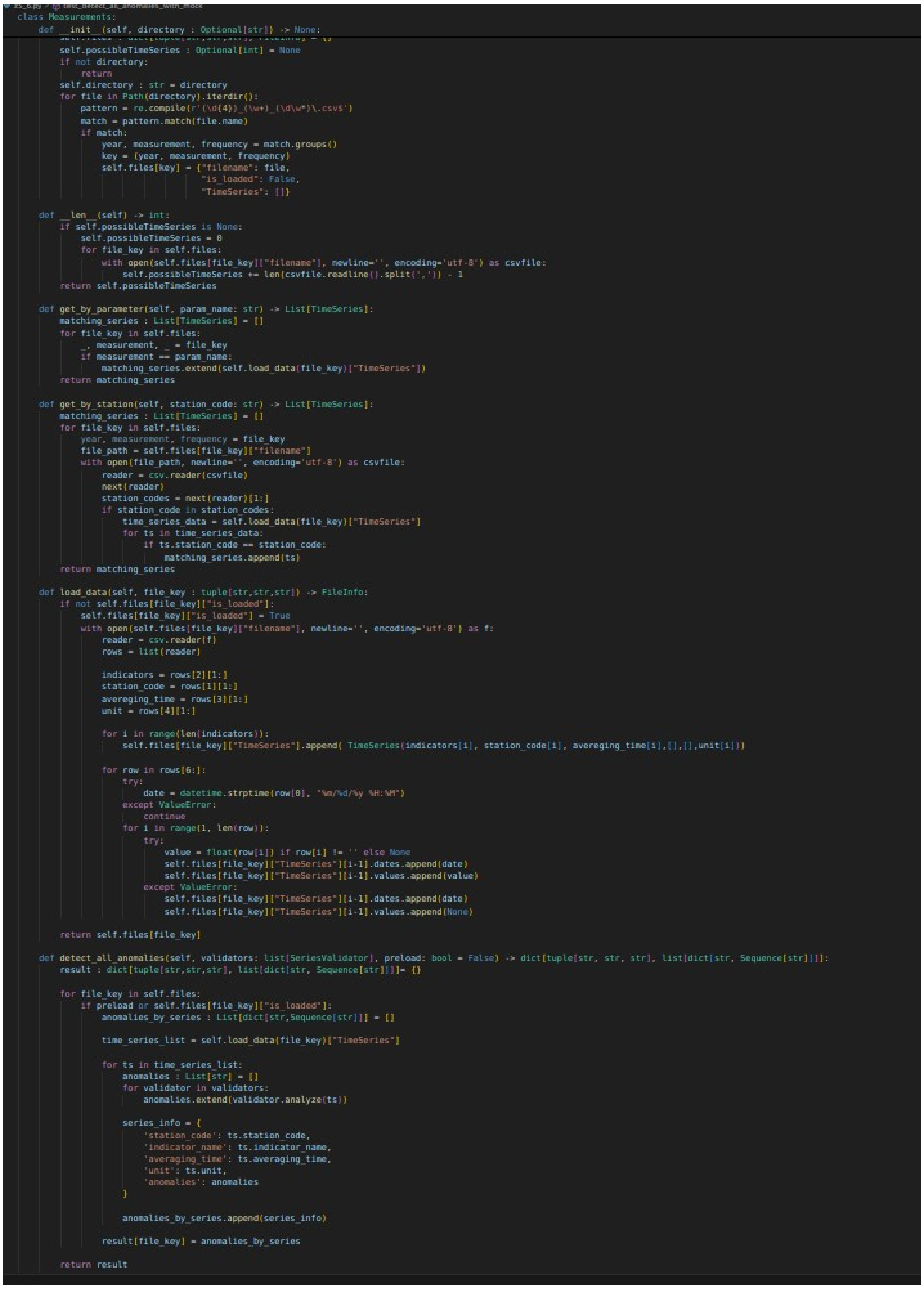
```python
import pytest


class Station:

    nr: str
    kod_stacji: str
    kod_miedz: str
    nazwa: str
    stary_kod: str
    data_begin: str
    data_end: str
    typ: str
    obszar: str
    rodzaj: str
    woj: str
    miejsc: str
    adres: str
    lon: str
    lat: str

    def __init__(
            self,
            nr : str,
            kod_stacji : str,
            kod_miedz : str,
            nazwa : str,
            stary_kod : str,
            data_begin : str,
            data_end : str,
            typ : str,
            obszar : str,
            rodzaj : str,
            woj : str,
            miejsc : str,
            adres : str,
            lon : str,
            lat : str
    )->None:
        for name, value in locals().items():
            if name != 'self':
                setattr(self, name, value)

        return

    def __str__(self) ->str:
        res : str = ''
        for k, v in self.__dict__.items():
            res += f'{k}: {v};'
        return res

    def __repr__(self) -> str:
        cls_name : str = self.__class__.__name__
        attrs : str = ', '.join(f"{k}={repr(v)}" for k, v in self.__dict__.items())
        return f"{cls_name}({attrs})"

    def __eq__(self, other: object) -> bool:
        if not isinstance(other, Station):
            return False
        return self.kod_stacji == other.kod_stacji


def test_station_eq():
    s1 = Station('',"ST01",'','','','','','','','','','','','','')
    s2 = Station('',"ST01",'','','','','','','','','','','','','')
    s3 = Station('',"ST02",'','','','','','','','','','','','','')
    assert s1 == s2, "Stacje o tych samych kodach powinny być równe"
    assert s1 != s3, "Stacje o różnych kodach nie powinny być równe"
```



```python
import numpy as np
from datetime import datetime, timedelta, date
from typing import Optional, Union
import pytest

class TimeSeries:
    def __init__(self, indicator_name : str, station_code : str, averaging_time : str, dates : list[datetime], values : list [Optional[float]], unit : str) -> None:
        self.indicator_name : str = indicator_name
        self.station_code :str = station_code
        self.averaging_time : str = averaging_time
        self.dates : list[datetime] = dates
        self.values : list [Optional[float]] = values
        self.unit : str = unit



    def __getitem__(self, key: Union[int, slice, datetime, date]) -> tuple[datetime, float | None] | list[tuple[datetime, float | None]]:
        if isinstance(key, slice):
            return [(self.dates[i], self.values[i]) for i in range(*key.indices(len(self.dates)))]
        elif isinstance(key, int):
            return self.dates[key], self.values[key]
        elif isinstance(key, (datetime, date)):
            if isinstance(key, date) and not isinstance(key, datetime):
                key = datetime.combine(key, datetime.min.time())
            if key in self.dates:
                idx : int  = self.dates.index(key)
                return self.dates[idx], self.values[idx]
            else:
                raise KeyError(f"Data {key} nie istnieje w danych pomiarowych.")
        else:
            raise TypeError(f"Nieobsługiwany typ klucza: {type(key)}")


    @property
    def mean(self) -> float | None:
        # Obliczanie średniej arytmetycznej
        values : list[float]= [v for v in self.values if v is not None]
        if values:
            return float(np.mean(values))
        return None

    @property
    def stddev(self) -> float | None:
        # Obliczanie odchylenia standardowego
        values : list[float] = [v for v in self.values if v is not None]
        if values:
            return float(np.std(values))
        return None
```

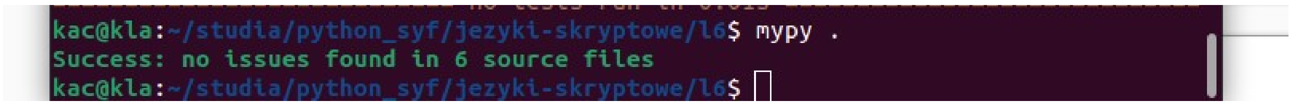


```python
from typing import List
from datetime import datetime, timedelta
import pytest

class SeriesValidator(ABC):
    @abstractmethod
    def analyze(self, series: TimeSeries) -> List[str]:
        pass


class OutlierDetector(SeriesValidator):
    def __init__(self, k: float) -> None:
        self.k : float = k

    def analyze(self, series: TimeSeries) -> List[str]:
        mean : float | None = series.mean
        stddev : float | None = series.stddev
        if mean is None or stddev is None:
            return []

        anomalies : List[str] = []
        for date, value in zip(series.dates, series.values):
            if value is not None and abs(value - mean) > self.k * stddev:
                anomalies.append(f"Outlier detected: {value} at {date} (more than {self.k} standard deviations from mean)")
        return anomalies


class ZeroSpikeDetector(SeriesValidator):
    def analyze(self, series: TimeSeries) -> List[str]:
        anomalies : List[str] = []
        count : int = 0
        for date, value in zip(series.dates, series.values):
            if value == 0 or value is None:
                count += 1
            else:
                count = 0

            if count >= 3:
                anomalies.append(f"Zero spike detected at {date} (3 or more consecutive zeros or missing values)")
        return anomalies



class ThresholdDetector(SeriesValidator):
    def __init__(self, threshold: float) -> None:
        self.threshold : float = threshold

    def analyze(self, series: TimeSeries) -> List[str]:
        anomalies : List[str] = []
        for date, value in zip(series.dates, series.values):
            if value is not None and value > self.threshold:
                anomalies.append(f"Threshold exceeded: {value} at {date} (greater than {self.threshold})")
        return anomalies


class CompositeValidator(SeriesValidator):
    def __init__(self, validators : List[SeriesValidator], mode: str = "OR") ->None:
        self.validators : List[SeriesValidator] = validators
        self.mode : str = mode.upper()

    def analyze(self, series: TimeSeries) -> List[str]:
        anomalies : List[str]=[]

        result : List[str]
        # AND returns only if every validator had result if not empty list
        if self.mode == "AND":
            for validator in self.validators:
                result = validator.analyze(series)
                if result:
                    anomalies.extend(result)
                else:
                    return []
        # OR returns everything that validators returns
        elif self.mode == "OR":
            for validator in self.validators:
                result = validator.analyze(series)
                anomalies.extend(result)
            return anomalies

        else:
            raise ValueError("Mode must be 'AND' or 'OR'")
        return []
```



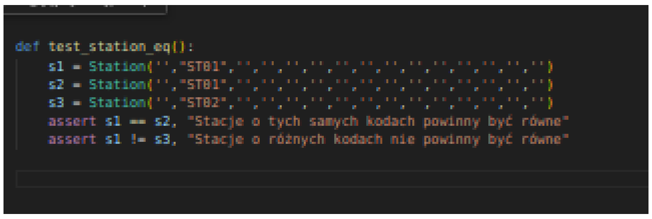
```python
class Measurements:
    def __init__(self, directory : Optional[str]) -> None:
        self.files : dict[tuple[str,str,str], FileInfo] = {}
        self.possibleTimeSeries : Optional[int] = None
        if not directory:
            return
        self.directory : str = directory
        for file in Path(directory).iterdir():
            pattern = re.compile(r'(\d{4})_(\w+)_(\d\w*)\.csv$')
            match = pattern.match(file.name)
            if match:
                year, measurement, frequency = match.groups()
                key = (year, measurement, frequency)
                self.files[key] = {"filename": file,
                                   "is_loaded": False,
                                   "TimeSeries": []}

    def __len__(self) -> int:
        if self.possibleTimeSeries is None:
            self.possibleTimeSeries = 0
            for file_key in self.files:
                with open(self.files[file_key]["filename"], newline='', encoding='utf-8') as csvfile:
                    self.possibleTimeSeries += len(csvfile.readline().split(',')) - 1
        return self.possibleTimeSeries

    def get_by_parameter(self, param_name: str) -> List[TimeSeries]:
        matching_series : List[TimeSeries] = []
        for file_key in self.files:
            _, measurement, _ = file_key
            if measurement == param_name:
                matching_series.extend(self.load_data(file_key)["TimeSeries"])
        return matching_series

    def get_by_station(self, station_code: str) -> List[TimeSeries]:
        matching_series : List[TimeSeries] = []
        for file_key in self.files:
            year, measurement, frequency = file_key
            file_path = self.files[file_key]["filename"]
            with open(file_path, newline='', encoding='utf-8') as csvfile:
                reader = csv.reader(csvfile)
                next(reader)
                station_codes = next(reader)[1:]
                if station_code in station_codes:
                    time_series_data = self.load_data(file_key)["TimeSeries"]
                    for ts in time_series_data:
                        if ts.station_code == station_code:
                            matching_series.append(ts)
        return matching_series

    def load_data(self, file_key : tuple[str,str,str]) -> FileInfo:
        if not self.files[file_key]["is_loaded"]:
            self.files[file_key]["is_loaded"] = True
            with open(self.files[file_key]["filename"], newline='', encoding='utf-8') as f:
                reader = csv.reader(f)
                rows = list(reader)

                indicators = rows[2][1:]
                station_code = rows[1][1:]
                avereging_time = rows[3][1:]
                unit = rows[4][1:]

                for i in range(len(indicators)):
                    self.files[file_key]["TimeSeries"].append( TimeSeries(indicators[i], station_code[i], avereging_time[i],[],[],unit[i]))

                for row in rows[6:]:
                    try:
                        date = datetime.strptime(row[0], "%m/%d/%y %H:%M")
                    except ValueError:
                        continue
                    for i in range(1, len(row)):
                        try:
                            value = float(row[i]) if row[i] != '' else None
                            self.files[file_key]["TimeSeries"][i-1].dates.append(date)
                            self.files[file_key]["TimeSeries"][i-1].values.append(value)
                        except ValueError:
                            self.files[file_key]["TimeSeries"][i-1].dates.append(date)
                            self.files[file_key]["TimeSeries"][i-1].values.append(None)

        return self.files[file_key]

    def detect_all_anomalies(self, validators: list[SeriesValidator], preload: bool = False) -> dict[tuple[str, str, str], list[dict[str, Sequence[str]]]]:
        result : dict[tuple[str,str,str], list[dict[str, Sequence[str]]]]= {}

        for file_key in self.files:
            if preload or self.files[file_key]["is_loaded"]:
                anomalies_by_series : List[dict[str,Sequence[str]]] = []

                time_series_list = self.load_data(file_key)["TimeSeries"]

                for ts in time_series_list:
                    anomalies : List[str] = []
                    for validator in validators:
                        anomalies.extend(validator.analyze(ts))

                    series_info = {
                        'station_code': ts.station_code,
                        'indicator_name': ts.indicator_name,
                        'averaging_time': ts.averaging_time,
                        'unit': ts.unit,
                        'anomalies': anomalies
                    }

                    anomalies_by_series.append(series_info)

                result[file_key] = anomalies_by_series

        return result
```

z2

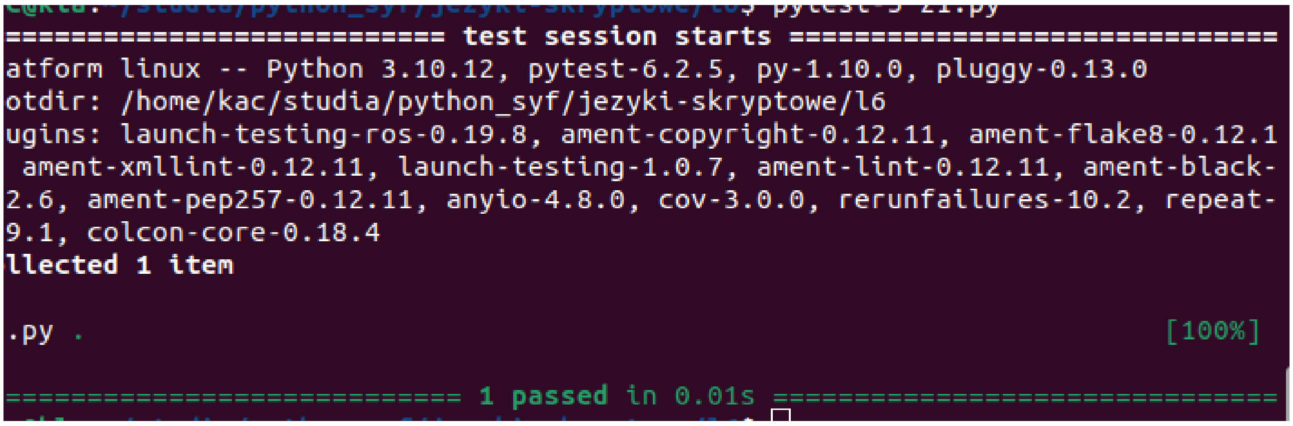


```
kac@kla:~/studia/python_syf/jezyki-skryptowe/l6$ mypy .
Success: no issues found in 6 source files
kac@kla:~/studia/python_syf/jezyki-skryptowe/l6$ 
```

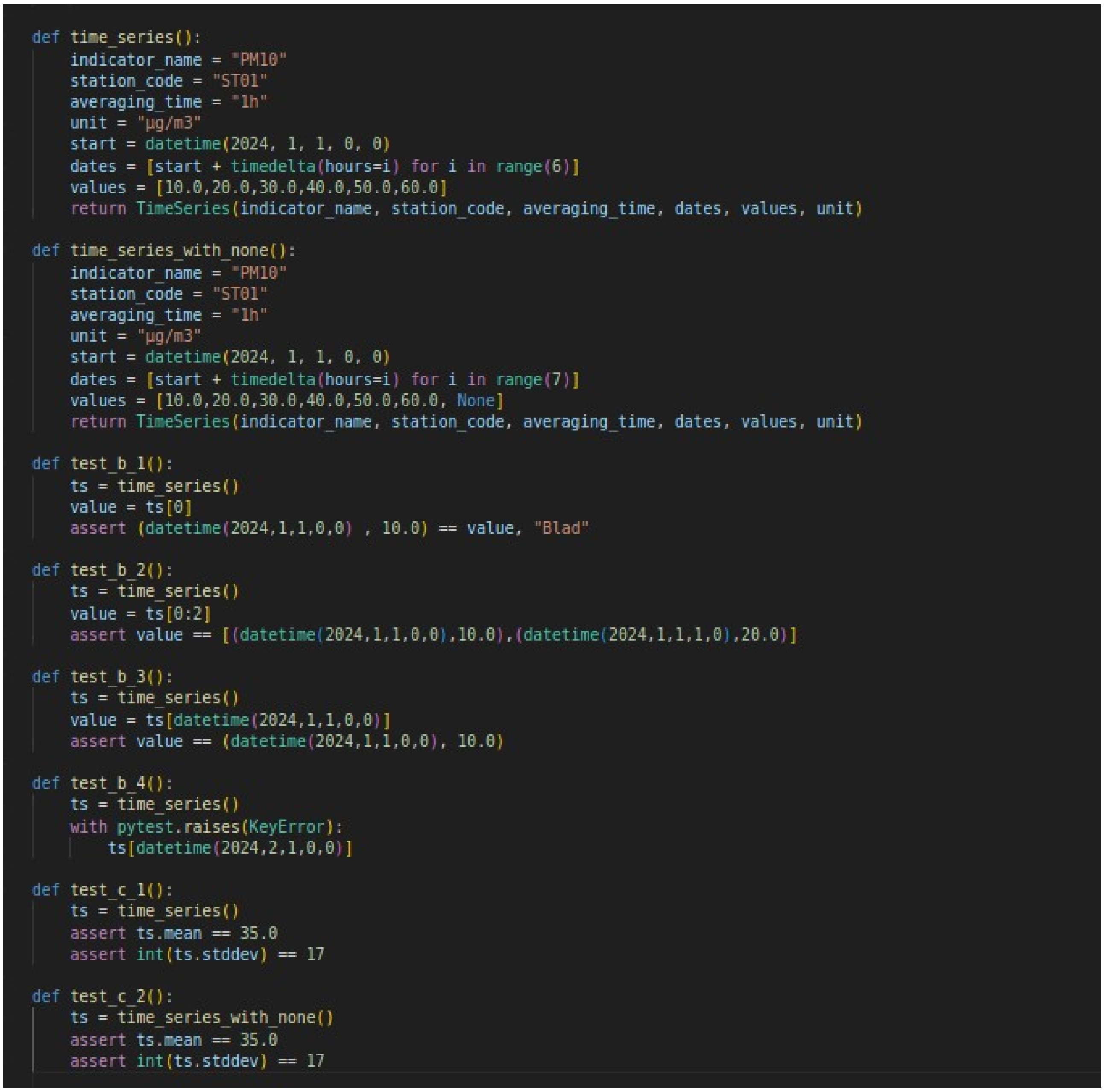
z3



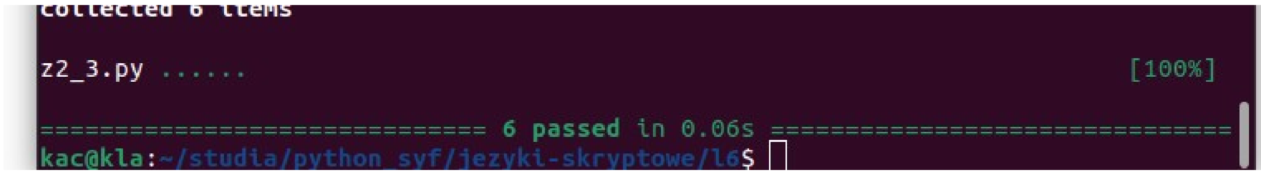
```
def test_station_eq():
    s1 = Station('',"ST01",'','','','','','','','','','','','','')
    s2 = Station('',"ST01",'','','','','','','','','','','','','')
    s3 = Station('',"ST02",'','','','','','','','','','','','','')
    assert s1 == s2, "Stacje o tych samych kodach powinny być równe"
    assert s1 != s3, "Stacje o różnych kodach nie powinny być równe"
```



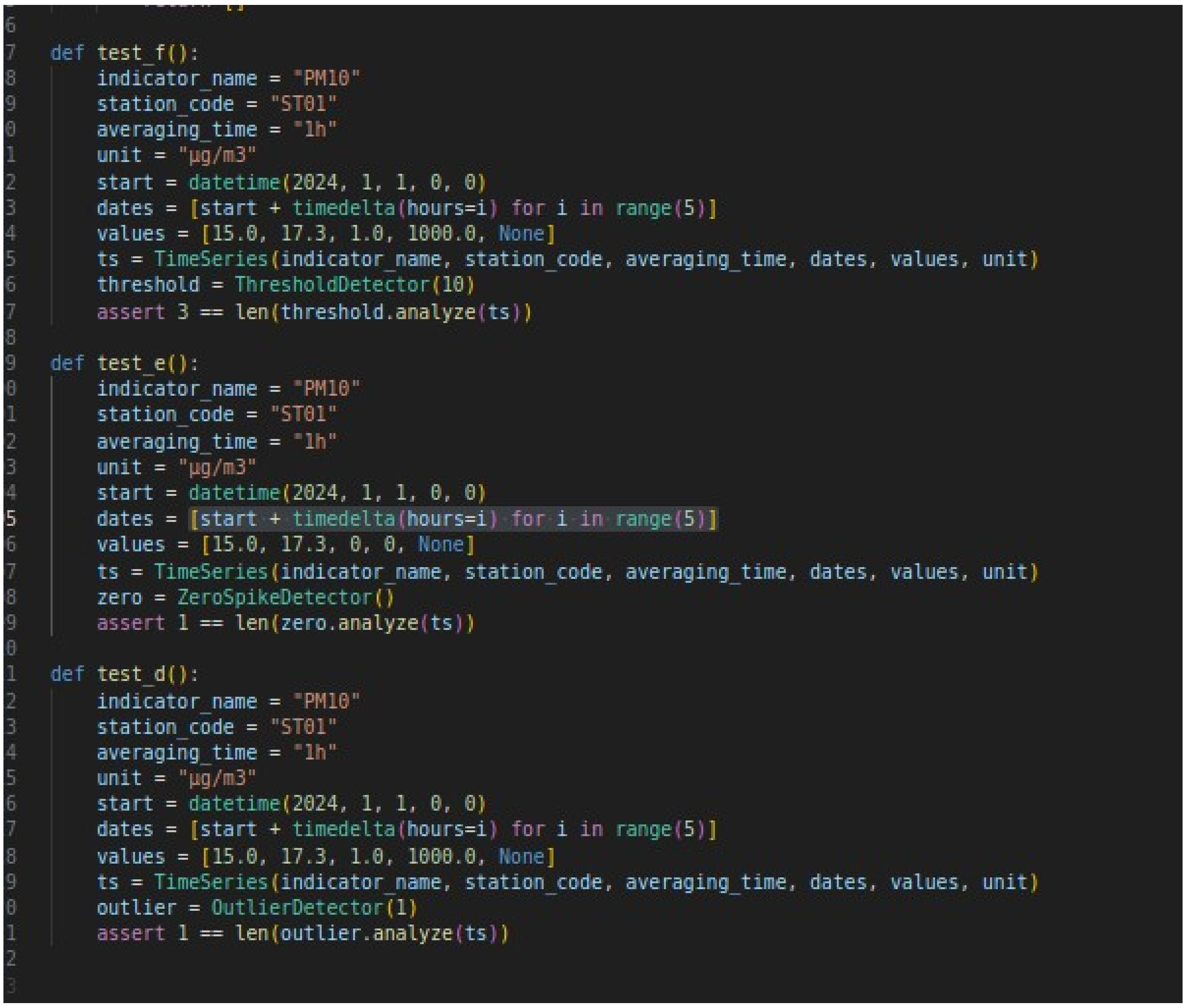
```
c@kta:~/studia/python_syf/jezyki-skryptowe/l6$ pytest -s z1.py
=========================== test session starts ================================
atform linux -- Python 3.10.12, pytest-6.2.5, py-1.10.0, pluggy-0.13.0
otdir: /home/kac/studia/python_syf/jezyki-skryptowe/l6
ugins: launch-testing-ros-0.19.8, ament-copyright-0.12.11, ament-flake8-0.12.1
 ament-xmllint-0.12.11, launch-testing-1.0.7, ament-lint-0.12.11, ament-black-
2.6, ament-pep257-0.12.11, anyio-4.8.0, cov-3.0.0, rerunfailures-10.2, repeat-
9.1, colcon-core-0.18.4
llected 1 item

.py .                                                                    [100%]

============================ 1 passed in 0.01s =================================
```



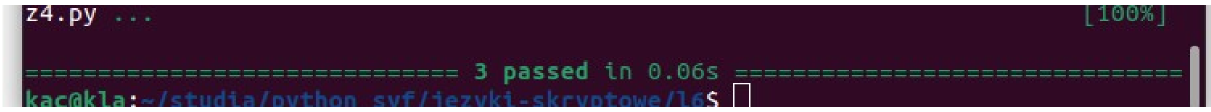
```python
def time_series():
    indicator_name = "PM10"
    station_code = "ST01"
    averaging_time = "1h"
    unit = "µg/m3"
    start = datetime(2024, 1, 1, 0, 0)
    dates = [start + timedelta(hours=i) for i in range(6)]
    values = [10.0,20.0,30.0,40.0,50.0,60.0]
    return TimeSeries(indicator_name, station_code, averaging_time, dates, values, unit)

def time_series_with_none():
    indicator_name = "PM10"
    station_code = "ST01"
    averaging_time = "1h"
    unit = "µg/m3"
    start = datetime(2024, 1, 1, 0, 0)
    dates = [start + timedelta(hours=i) for i in range(7)]
    values = [10.0,20.0,30.0,40.0,50.0,60.0, None]
    return TimeSeries(indicator_name, station_code, averaging_time, dates, values, unit)

def test_b_1():
    ts = time_series()
    value = ts[0]
    assert (datetime(2024,1,1,0,0) , 10.0) == value, "Blad"

def test_b_2():
    ts = time_series()
    value = ts[0:2]
    assert value == [(datetime(2024,1,1,0,0),10.0),(datetime(2024,1,1,1,0),20.0)]

def test_b_3():
    ts = time_series()
    value = ts[datetime(2024,1,1,0,0)]
    assert value == (datetime(2024,1,1,0,0), 10.0)

def test_b_4():
    ts = time_series()
    with pytest.raises(KeyError):
        ts[datetime(2024,2,1,0,0)]

def test_c_1():
    ts = time_series()
    assert ts.mean == 35.0
    assert int(ts.stddev) == 17

def test_c_2():
    ts = time_series_with_none()
    assert ts.mean == 35.0
    assert int(ts.stddev) == 17
```



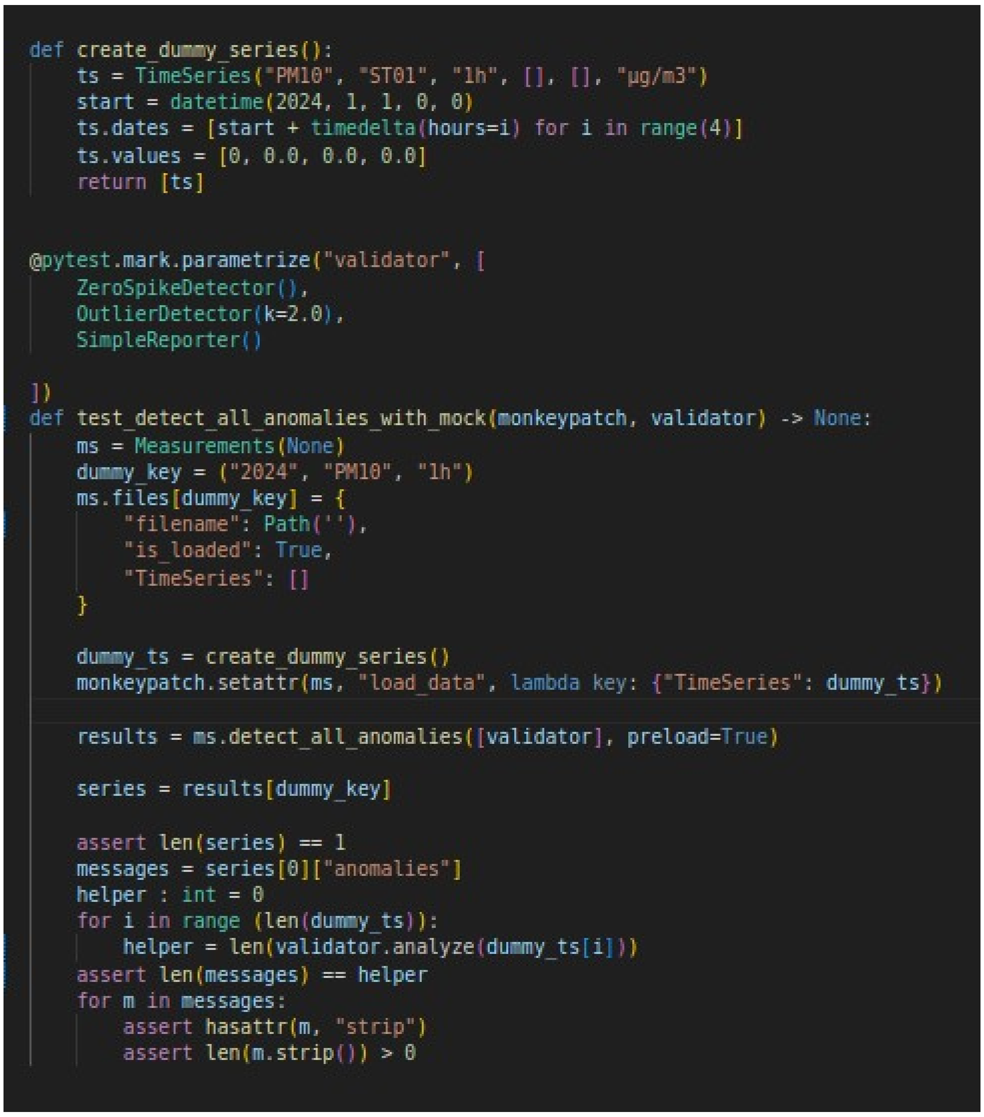
```
collected 6 items

z2_3.py ......                                                       [100%]

=============================== 6 passed in 0.06s ================================
kac@kla:~/studia/python_syf/jezyki-skryptowe/l6$
```



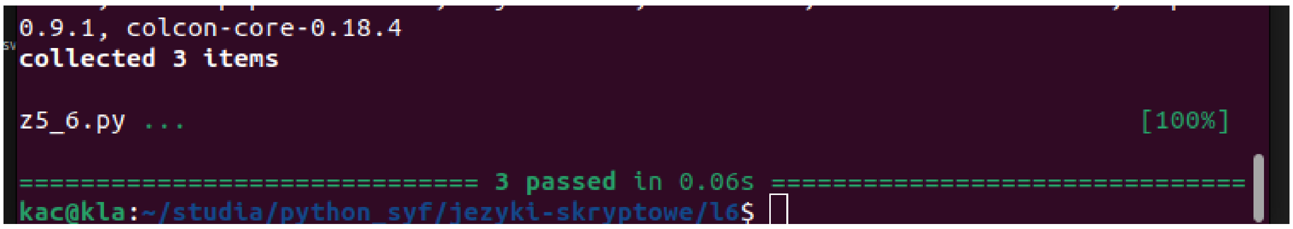
```python
def test_f():
    indicator_name = "PM10"
    station_code = "ST01"
    averaging_time = "1h"
    unit = "µg/m3"
    start = datetime(2024, 1, 1, 0, 0)
    dates = [start + timedelta(hours=i) for i in range(5)]
    values = [15.0, 17.3, 1.0, 1000.0, None]
    ts = TimeSeries(indicator_name, station_code, averaging_time, dates, values, unit)
    threshold = ThresholdDetector(10)
    assert 3 == len(threshold.analyze(ts))

def test_e():
    indicator_name = "PM10"
    station_code = "ST01"
    averaging_time = "1h"
    unit = "µg/m3"
    start = datetime(2024, 1, 1, 0, 0)
    dates = [start + timedelta(hours=i) for i in range(5)]
    values = [15.0, 17.3, 0, 0, None]
    ts = TimeSeries(indicator_name, station_code, averaging_time, dates, values, unit)
    zero = ZeroSpikeDetector()
    assert 1 == len(zero.analyze(ts))

def test_d():
    indicator_name = "PM10"
    station_code = "ST01"
    averaging_time = "1h"
    unit = "µg/m3"
    start = datetime(2024, 1, 1, 0, 0)
    dates = [start + timedelta(hours=i) for i in range(5)]
    values = [15.0, 17.3, 1.0, 1000.0, None]
    ts = TimeSeries(indicator_name, station_code, averaging_time, dates, values, unit)
    outlier = OutlierDetector(1)
    assert 1 == len(outlier.analyze(ts))
```



```
z4.py ...                                                                  [100%]

============================== 3 passed in 0.06s ===============================
kac@kla:~/studia/python_syf/jezyki-skryptowe/l6$ 
```



```python
def create_dummy_series():
    ts = TimeSeries("PM10", "ST01", "1h", [], [], "µg/m3")
    start = datetime(2024, 1, 1, 0, 0)
    ts.dates = [start + timedelta(hours=i) for i in range(4)]
    ts.values = [0, 0.0, 0.0, 0.0]
    return [ts]


@pytest.mark.parametrize("validator", [
    ZeroSpikeDetector(),
    OutlierDetector(k=2.0),
    SimpleReporter()

])
def test_detect_all_anomalies_with_mock(monkeypatch, validator) -> None:
    ms = Measurements(None)
    dummy_key = ("2024", "PM10", "1h")
    ms.files[dummy_key] = {
        "filename": Path(''),
        "is_loaded": True,
        "TimeSeries": []
    }

    dummy_ts = create_dummy_series()
    monkeypatch.setattr(ms, "load_data", lambda key: {"TimeSeries": dummy_ts})

    results = ms.detect_all_anomalies([validator], preload=True)

    series = results[dummy_key]

    assert len(series) == 1
    messages = series[0]["anomalies"]
    helper : int = 0
    for i in range (len(dummy_ts)):
        helper = len(validator.analyze(dummy_ts[i]))
    assert len(messages) == helper
    for m in messages:
        assert hasattr(m, "strip")
        assert len(m.strip()) > 0
```



```
0.9.1, colcon-core-0.18.4
collected 3 items

z5_6.py ...                                                          [100%]

============================== 3 passed in 0.06s ===============================
kac@kla:~/studia/python_syf/jezyki-skryptowe/l6$ 
```