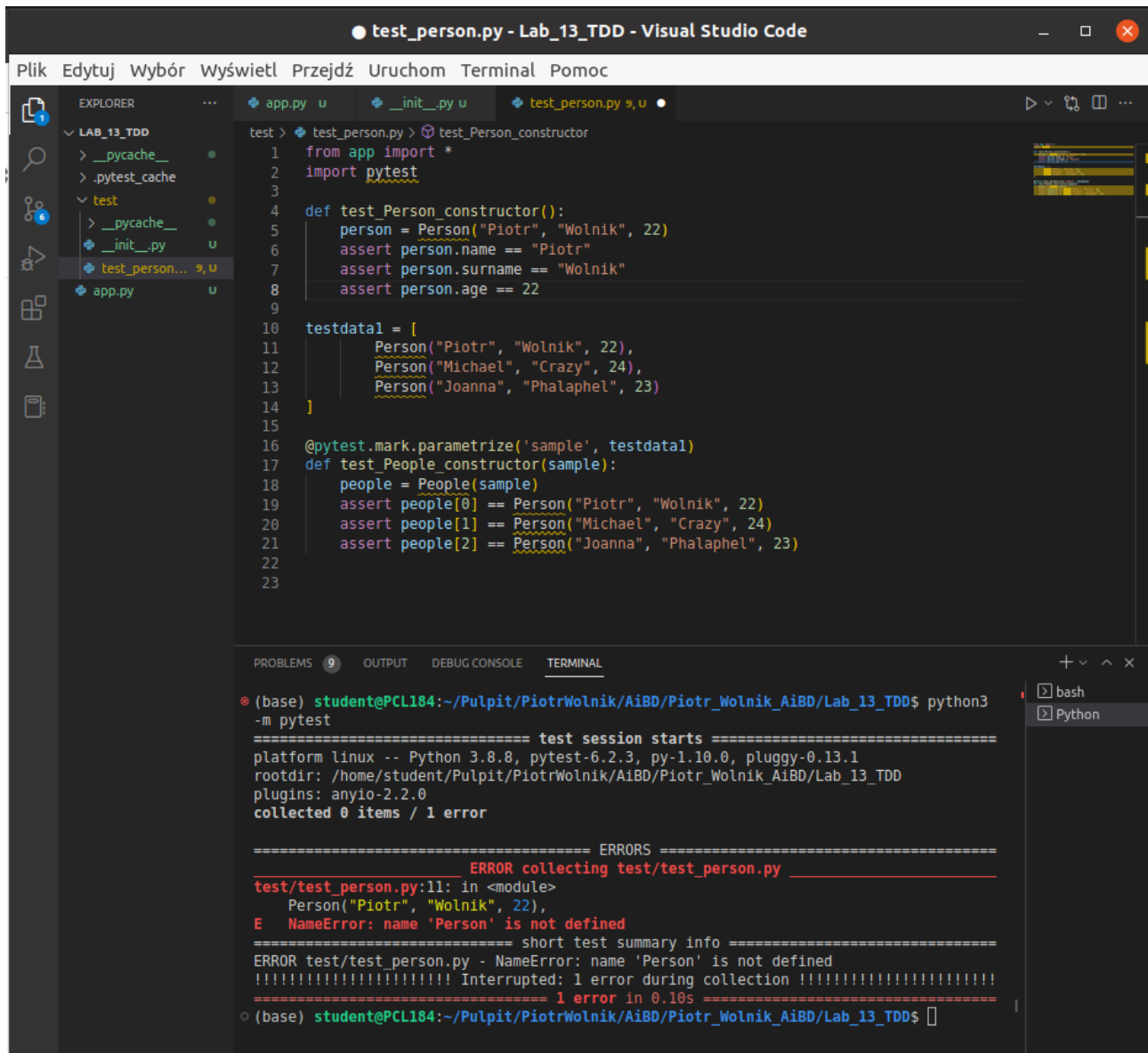


In my case I decided to use TDD on problem that was involving creating class `Person` and `People`. The first one, is supposed to represent a structure which attributes are: first name, second name and age. We have possibility of comparing objects of that class and printing in a special format it's data. Second class is working like a container for `Person` class objects'. Options such as getting access to specific element and printing on the console are available.

Firstly, I started by creating simple tests that are described below (no implementation at that point was provided):



The screenshot shows the Visual Studio Code interface with a file named `test_person.py` open. The code defines two test functions: `test_Person_constructor` and `test_People_constructor`. The first test creates a `Person` object and asserts its attributes. The second test creates a `People` object containing a list of `Person` objects and asserts the list's contents. The terminal output shows the test session starting, but it fails with a `NameError: name 'Person' is not defined` because the `Person` class is not imported or defined in the test file's scope.

```
test > test_person.py > test_Person_constructor
1 from app import *
2 import pytest
3
4 def test_Person_constructor():
5     person = Person("Piotr", "Wolnik", 22)
6     assert person.name == "Piotr"
7     assert person.surname == "Wolnik"
8     assert person.age == 22
9
10 testdata = [
11     Person("Piotr", "Wolnik", 22),
12     Person("Michael", "Crazy", 24),
13     Person("Joanna", "Phalaphel", 23)
14 ]
15
16 @pytest.mark.parametrize('sample', testdata)
17 def test_People_constructor(sample):
18     people = People(sample)
19     assert people[0] == Person("Piotr", "Wolnik", 22)
20     assert people[1] == Person("Michael", "Crazy", 24)
21     assert people[2] == Person("Joanna", "Phalaphel", 23)
22
23
```

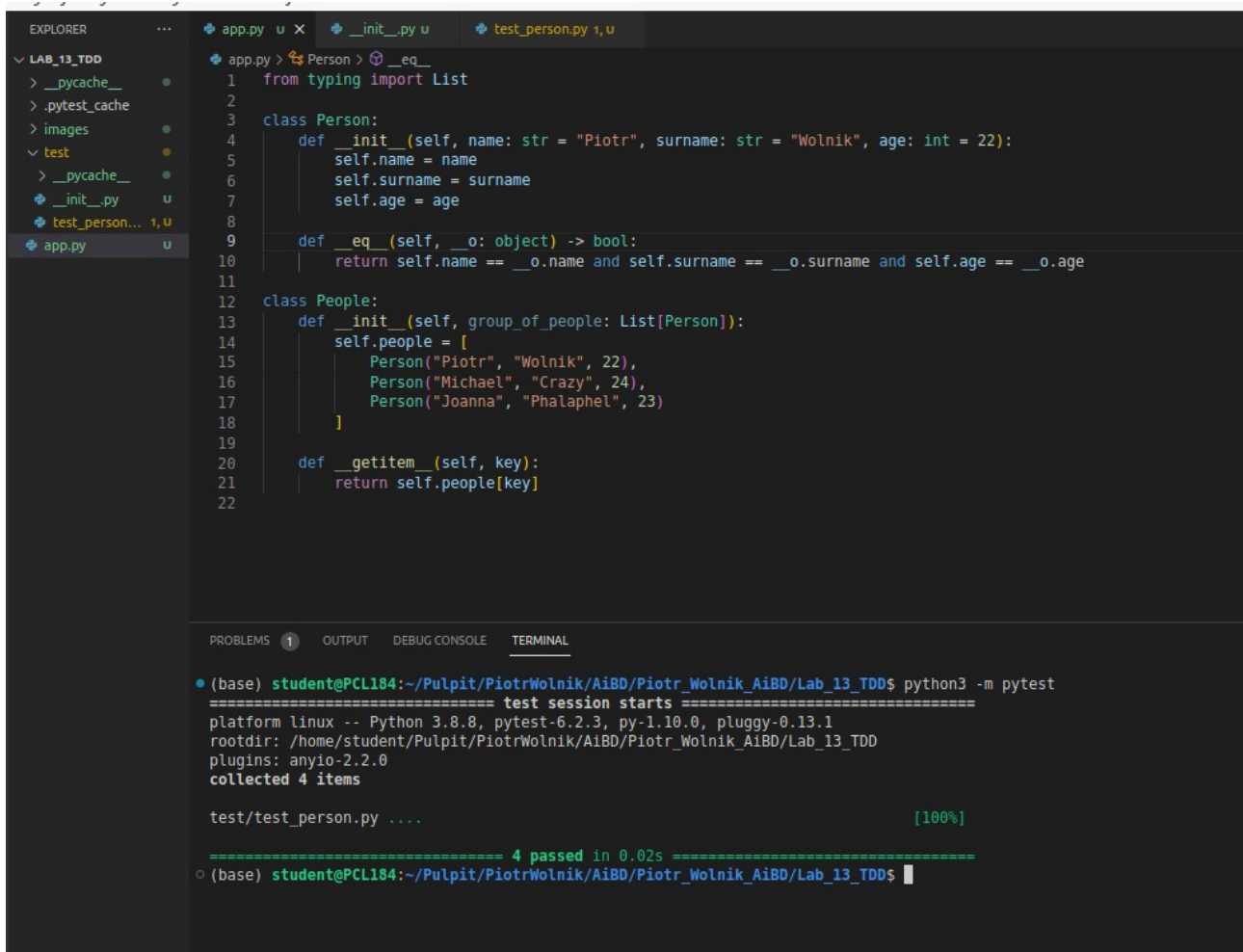
```
(base) student@PCL184:~/Pulpit/PiotrWolnik/AiBD/Piotr_Wolnik_AiBD/Lab_13_TDD$ python3
-m pytest
===== test session starts =====
platform linux -- Python 3.8.8, pytest-6.2.3, py-1.10.0, pluggy-0.13.1
rootdir: /home/student/Pulpit/PiotrWolnik/AiBD/Piotr_Wolnik_AiBD/Lab_13_TDD
plugins: anyio-2.2.0
collected 0 items / 1 error

===== ERRORS =====
test/test_person.py:11: in <module>
    Person("Piotr", "Wolnik", 22),
E   NameError: name 'Person' is not defined

===== short test summary info =====
ERROR test/test_person.py - NameError: name 'Person' is not defined
!!!!!!!!!!!!!!!!!!!! Interrupted: 1 error during collection !!!!!!!!!!!!!!!!!!!!!
===== 1 error in 0.10s =====
(base) student@PCL184:~/Pulpit/PiotrWolnik/AiBD/Piotr_Wolnik_AiBD/Lab_13_TDD$
```

As we can see we test constructors of both class to check if objects are created in a proper way. The second test involves using undefined index operator.

Then, we go straight to phase GREEN which stands for writing minimal implementation so that the tests would pass:



```
EXPLORER
  LAB_13_TDD
    > __pycache__
    > .pytest_cache
    > images
    > test
      > __pycache__
      > __init__.py
      > test_person.py
    > app.py

app.py
1 from typing import List
2
3 class Person:
4     def __init__(self, name: str = "Piotr", surname: str = "Wolnik", age: int = 22):
5         self.name = name
6         self.surname = surname
7         self.age = age
8
9     def __eq__(self, __o: object) -> bool:
10         return self.name == __o.name and self.surname == __o.surname and self.age == __o.age
11
12 class People:
13     def __init__(self, group_of_people: List[Person]):
14         self.people = [
15             Person("Piotr", "Wolnik", 22),
16             Person("Michael", "Crazy", 24),
17             Person("Joanna", "Phalaphel", 23)
18         ]
19
20     def __getitem__(self, key):
21         return self.people[key]
22
test_person.py
1 import pytest
2
3 @pytest.mark.parametrize('sample', testdata)
4 def test_str_method(sample):
5     people = People([Person("Piotr", "Wolnik", 22), Person("Michael", "Crazy", 24)])
6     assert str(people[0]) == sample[0]
7
8 testdata = [
9     ('Wolnik, Piotr -> 22'),
10    ('Crazy, Michael -> 24')
11 ]

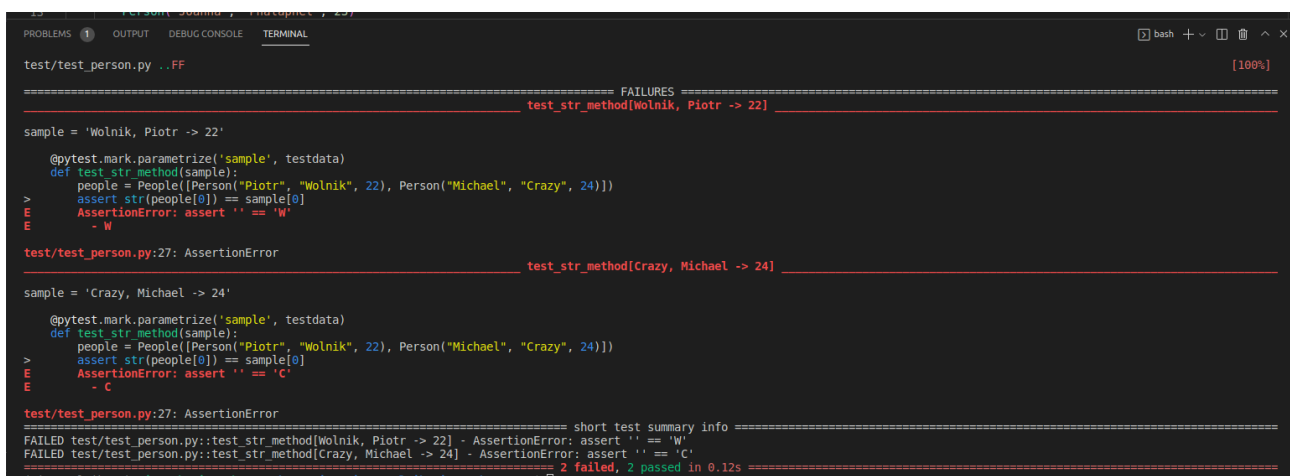
TERMINAL
(base) student@PCL184:~/Pulpit/PiotrWolnik/AiBD/Piotr_Wolnik_AiBD/Lab_13_TDD$ python3 -m pytest
===== test session starts =====
platform linux -- Python 3.8.8, pytest-6.2.3, py-1.10.0, pluggy-0.13.1
rootdir: /home/student/Pulpit/PiotrWolnik/AiBD/Piotr_Wolnik_AiBD/Lab_13_TDD
plugins: anyio-2.2.0
collected 4 items

test/test_person.py .... [100%]

===== 4 passed in 0.02s =====
(base) student@PCL184:~/Pulpit/PiotrWolnik/AiBD/Piotr_Wolnik_AiBD/Lab_13_TDD$
```

All tests pass.

Next, we provide some new test that is responsible for checking if `__str__` function of `Person` class is working correctly. Of course at this point we provide no implementation of that function either (we just define it):



```
test/test_person.py:27: AssertionError
===== FAILURES =====
test_str_method[Wolnik, Piotr -> 22]

sample = 'Wolnik, Piotr -> 22'

@pytest.mark.parametrize('sample', testdata)
def test_str_method(sample):
    people = People([Person("Piotr", "Wolnik", 22), Person("Michael", "Crazy", 24)])
    > assert str(people[0]) == sample[0]
E   AssertionError: assert '' == 'W'
E   - W

test/test_person.py:27: AssertionError
test_str_method[Crazy, Michael -> 24]

sample = 'Crazy, Michael -> 24'

@pytest.mark.parametrize('sample', testdata)
def test_str_method(sample):
    people = People([Person("Piotr", "Wolnik", 22), Person("Michael", "Crazy", 24)])
    > assert str(people[0]) == sample[0]
E   AssertionError: assert '' == 'C'
E   - C

test/test_person.py:27: AssertionError

===== short test summary info =====
FAILED test/test_person.py::test_str_method[Wolnik, Piotr -> 22] - AssertionError: assert '' == 'W'
FAILED test/test_person.py::test_str_method[Crazy, Michael -> 24] - AssertionError: assert '' == 'C'
===== 2 failed, 2 passed in 0.12s =====
(niotrek@niotrek:~/Desktop/AiBD/Piotr_Wolnik_AiBD/Lab_13_TDD$
```

As we can see, test fails.

The overall cycle of TDD development is met.

Nevertheless, we want to have no tests that failed, so we provide implementation for `__str__` function of `Person` class with the intention that all assertions are correct:

```
21
22
23 testdata = ["Wolnik, Piotr -> 22", "Crazy, Michael -> 24"]
24 @pytest.mark.parametrize('sample', testdata)
25 def test_str_method(sample):
26     people = People([Person("Piotr", "Wolnik", 22), Person("Michael", "Crazy", 24)])
27     assert str(people[0]) == testdata[0]
28     assert str(people[1]) == testdata[1]
29
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL

(piotrWolnikEnv) piotrek@piotrek:~/Desktop/AiBD/Piotr\_Wolnik\_AiBD/Lab\_13\_TDD\$ python3 -m pytest

===== test session starts =====

platform linux -- Python 3.10.8, pytest-7.1.2, pluggy-1.0.0  
rootdir: /home/piotrek/Desktop/AiBD/Piotr\_Wolnik\_AiBD/Lab\_13\_TDD  
plugins: anyio-3.5.0  
collected 4 items

test/test\_person.py .... [100%]

===== 4 passed in 0.02s =====

(piotrWolnikEnv) piotrek@piotrek:~/Desktop/AiBD/Piotr\_Wolnik\_AiBD/Lab\_13\_TDD\$