Lab 5 - TDD & Pair Programming

Introduction

In this lab you will try "hands on" two coding approaches learned this week:

- Test-Driven Development (TDD)
- Pair Programming

For the exercise divide yourself into pairs of students. Each pair will submit a series of 3 files at the end of the lab: 3 tests files & 3 coding files. Each <test_i, code_i> pair will represent the relevant code (and the matching test) for the i'th stage (1<= i <= 3).

If the group is uneven, then there will be exactly one student without a pair – which will need to write both the code & the tests. In that case the student will submit only the first 2 test files & first 2 code files.

One possible platform is Google collab, with each pair in a separate breakout room. At the end of the lab the collab files will be submitted for a Magen grade.

Note - you are required only basic documentation of the code, and only for non-trivial parts. Also, **pytest** is not mandatory.

The Problem

You will be given three incremental versions of the functions (each function in a different file) of the "FizzBuzz" math game. Each version is more advanced than the version from the previous stage.

You are required to program in pairs – so that one partner initially writes a test file (named *test_i.py* for 1<= i <= 3), which is supposed to PASS (all ASSERTs are "true") only if the given function *fizzBuzz_i (num)* (in the *code_i.py* file) is correct. If the *fizzBuzz_i (num)* function is incorrect – the test is supposed to FAIL.

After each test file *test_i.py* is written – the second student in the pair should adjust the code in the *code_i.py* file with the *fizzBuzz_i (num)* function so it will PASS the given test, written by the first student.

After finishing both *test_i.py* and *code_i.py* files for stage *i*, save both files and continue to stage *i+1* which will be implemented in two new files.

Below is a description of the 3 different *fizzBuzz_i (num)* functions, each in the *code_i.py* file.

Stage 1

Write a function *fizzBuzz_1(num)* function which receives an int *num* and returns:

- "Fizz" - if the number *num* is a multiple of **3**
- "Buzz" - if the number *num* is a multiple of **5**
- "FizzBuzz" - if the number *num* is a multiple of both **5** and **3**
- *num* - **otherwise**

For example:

- fizzBuzz_1(3) -> "Fizz"
- fizzBuzz_1(5) -> "Buzz"
- fizzBuzz_1(300) -> "FizzBuzz"
- fizzBuzz_1(4) -> 4

At the end of this stage save the files *code_1.py* (with the *fizzBuzz_1(num)* function) and the *test_1.py*, which will be submitted at the end of the lab.

Stage 2

Write a function *fizzBuzz_2(num)* function which receives an int *num* and returns:

- "Fizz" - if the number *num* is a multiple of **3** or it has a **3** in it
- "Buzz" - if the number *num* is a multiple of **5** or it has a **5** in it
- "FizzBuzz" - if the number *num* is "Fizz" **and** "Buzz"
- *num* - **otherwise**

For example:

- fizzBuzz_2(3), fizzBuzz_2(23) -> "Fizz"
- fizzBuzz_2(5), fizzBuzz_2(52) -> "Buzz"
- fizzBuzz_2(300), fizzBuzz_2(513) -> "FizzBuzz"
- fizzBuzz_2(4) -> 4

At the end of this stage save the files *code_2.py* (with the *fizzBuzz_2(num)* function) and the *test_2.py*, which will be submitted at the end of the lab.

Stage 3

Write a function *fizzBuzz_3(num)* which extends the function *fizzBuzz_2(num)* in the following way: the function receives an int *num* and returns a string value based on the **even/uneven** number of reasons for *num* to have a "fizzBuzz" value.

If for a given number *num* there is an **even** number of reasons for it to be "fizz" (e.g. the number *num* is a multiple of **3** and it has a **3** in it) they will cancel each other and the output will be "num" of "Buzz" (if there is an **uneven** number of reason for it to be "Buzz"). If there is an **uneven** number of reasons for *num* to be "fizz" – then the output will be "fizz". The same goes for the "Buzz" output: and output will be "Buzz" only if there is an **uneven** number of reasons for it to be "Buzz".

Note that if *num* can be either "Fizz" or "Buzz" than the output is "FizzBuzz", and it is a case when there are both **uneven** reasons for *num* to be "Fizz" and **uneven** reasons for *num* to be "Buzz".

For example:

- fizzBuzz_3(3) -> 3
  (because it is "Fizz" both due to ending with 3 and being divisible by 3)

- fizzBuzz_3(13) -> "Fizz"
  (because it is "Fizz" both due to ending with 3 and being divisible by 3)

- fizzBuzz_3(25) -> 25
  (because it is "Buzz" both due to ending with 5 and being divisible by 5)

- fizzBuzz_3(7) -> 7

At the end of this stage save the files *code_3.py* (with the *fizzBuzz_3(num)* function) and the *test_3.py*, which will be submitted at the end of the lab.