

Desenvolvimento guiado por testes

Jailma Januário da Silva

Leonardo Massayuki Takuno

Resumo

Os objetivos desta parte são: (I) Introduzir conceitos de desenvolvimentos guiados por testes. (II) Entender o ciclo red, green e refactor. (III) Realizar um exemplo de testes guiados por testes.

Introdução

Este texto apresenta uma técnica utilizada para desenvolvimento de software, chamado de desenvolvimento guiado por testes (do inglês, Test Driven Development - TDD). O termo TDD é mais conhecido na comunidade de desenvolvimento, e por este motivo, o texto utilizará desta sigla daqui para frente.

O TDD é um estilo de desenvolvimento de software que tem como princípio construir testes de unidade antes de se construir o código. Obviamente, que ao fazer isso, o teste falha, pois o código nem existe. Então o programador deve, necessariamente, corrigir a falha. A construção do código se dá pela construção de vários casos de testes, seguindo as mesmas etapas de: criar um teste antes do código, verificar a falha e depois corrigir.

Após a correção do código, e com os testes executados corretamente, inicia-se a etapa de refatoração, que inclui atividades para melhoria do código, legibilidade e modularização, caso seja necessário.

Este processo é conhecido no TDD como “*red, green e refactor*”, a Figura 4.1 apresenta o ciclo.

Figura 4.1. Ciclo *red, green e refactor*

<u>fizzbuzz(n)</u>		
#caso de teste	Parâmetros de entrada	Resultado esperado
CT0001	n=1	<u>1</u>
CT0002	n=2	<u>2</u>
CT0003	n=3	<u>Fizz</u>
CT0004	n=4	<u>4</u>
CT0005	n=5	<u>Buzz</u>
CT0006	n=6	<u>Fizz</u>
CT0007	n=7	<u>7</u>
CT0008	n=8	<u>8</u>
CT0009	n=9	<u>Fizz</u>
CT0010	n=10	<u>Buzz</u>

Fonte: do autor, 2022.

Segundo Sale (2014), o TDD faz parte do processo de desenvolvimento ágil, que defende o desenvolvimento iterativo sobre um processo mais restritivo, tal como o processo cascata. Por esta razão, os testes são realizados o quanto antes, e não no final do processo de desenvolvimento. Esta parte apresenta como você pode aplicar TDD no desenvolvimento de software, e ainda, discute as vantagens de se trabalhar desta maneira.

Motivos para realizar TDD

O objetivo do TDD é dar ao desenvolvedor e aos stakeholders confiança na construção do código da aplicação que será entregue. A qualidade deve-se ao fato de que o desenvolvedor de software é forçado a pensar sobre o

problema, pois ele tem que cobrir todos os casos de testes para provar que o programa executa da maneira esperada.

Como visto anteriormente, o ciclo de TDD consiste em escrever um teste que falha, antes de escrever o código. Isto parece um tanto confuso, mas ficará mais claro ao passo que se executa a técnica de desenvolvimento. Essencialmente, os testes devem guiar o desenvolvimento de maneira, que ao falharem, eles permitem que o programador possa escrever um pedaço de código para tratar aquele caso de teste.

Por exemplo, se o desenvolvedor necessita de uma classe para representar algum tipo de dado, e também, métodos para acessar esses dados. Então o TDD sugere criar um método que teste a criação dessa classe. E ainda, executar o método de acesso, como se a classe e o método já existissem. Então o teste vai falhar indicando que a classe e o método não existem. Em seguida, o desenvolvedor deve corrigir a falha e construir a classe o método para o teste executar corretamente. Assim, o teste guia o desenvolvimento, fazendo o programador a pensar no comportamento e na responsabilidade, antes mesmo de escrever o código.

Escrever o teste antes de escrever o código é a atividade principal desta técnica, e ela tem vários propósitos:

- Escrever testes ajudam a pensar no problema que deseja solucionar;
- Ajudam a pensar em casos de exceção;
 - E se a função está aguardando um inteiro e ela recebe como parâmetro de entrada uma string?
 - E se a função está aguardando números inteiros positivos e recebe algum número negativo?
 - Nestes casos, o que fazer? O sistema deve gerar exceção? Qual exceção?

Estas são algumas questões que surgem enquanto se escreve testes para uma pequena funcionalidade. Pode-se perceber o quanto de testes são necessários para cobrir os diferentes cenários de testes que podem surgir.

Vantagens TDD

Uma das principais vantagens de se usar TDD no projeto reside na garantia de realizar testes no código enquanto as funcionalidades estão sendo desenvolvidas, ao invés de deixar os testes para o final do projeto. Contudo, segundo Sale (2014) utilizar TDD traz alguns outros benefícios que incluem:

- Enquanto o software evolui, é possível incluir cobertura de testes em todas as funcionalidades da aplicação.
- Pode-se confiar no código produzido, pois os testes provam que o código comporta-se da maneira esperada.
- Falhar rapidamente! Um conjunto de testes permite que se identifiquem bugs conforme eles são criados, ao invés de ter que encontrar mais tarde nos testes de integração, ou pior ainda, em produção.

Escreva o teste para o código a ser desenvolvido, observe falhar, escreva o código, veja o código passar nos testes, e adicione mais testes.

Escrevendo código guiado por testes

Esta seção apresenta uma aplicação prática do TDD sobre um problema conhecido pelos desenvolvedores de software denominado por FizzBuzz. Este problema consiste em exibir uma lista de números de 1 até 100, um em cada linha, com as seguintes condições:

- Números divisíveis por 3 devem aparecer como "Fizz" ao invés do número;

- Números divisíveis por 5 devem aparecer como "Buzz" ao invés do número;
- Números divisíveis por 3 e por 5 devem aparecer como "FizzBuzz" ao invés do número;

Para este problema considere um plano de testes conforme apresenta a Tabela 4.1. Esse plano de teste contém alguns casos de testes iniciais para que se possa pensar nas funções para o tratamento desses casos de testes.

Tabela 4.1. Planejando os casos de testes

<u>fizzbuzz(n)</u>		
#caso de teste	Parâmetros de entrada	Resultado esperado
CT0001	n=1	<u>1</u>
CT0002	n=2	<u>2</u>
CT0003	n=3	<u>Fizz</u>
CT0004	n=4	<u>4</u>
CT0005	n=5	<u>Buzz</u>
CT0006	n=6	<u>Fizz</u>
CT0007	n=7	<u>7</u>
CT0008	n=8	<u>8</u>
CT0009	n=9	<u>Fizz</u>
CT0010	n=10	<u>Buzz</u>

Fonte: do autor, 2022.

Para cada caso de teste é necessário criar uma função, neste exemplo, será utilizado o Pytest como arcabouço, conforme apresenta o arquivo test_fizzbuzz.py como ilustra a Codificação 4.1.

Codificação 4.1. test_fizzbuzz.py

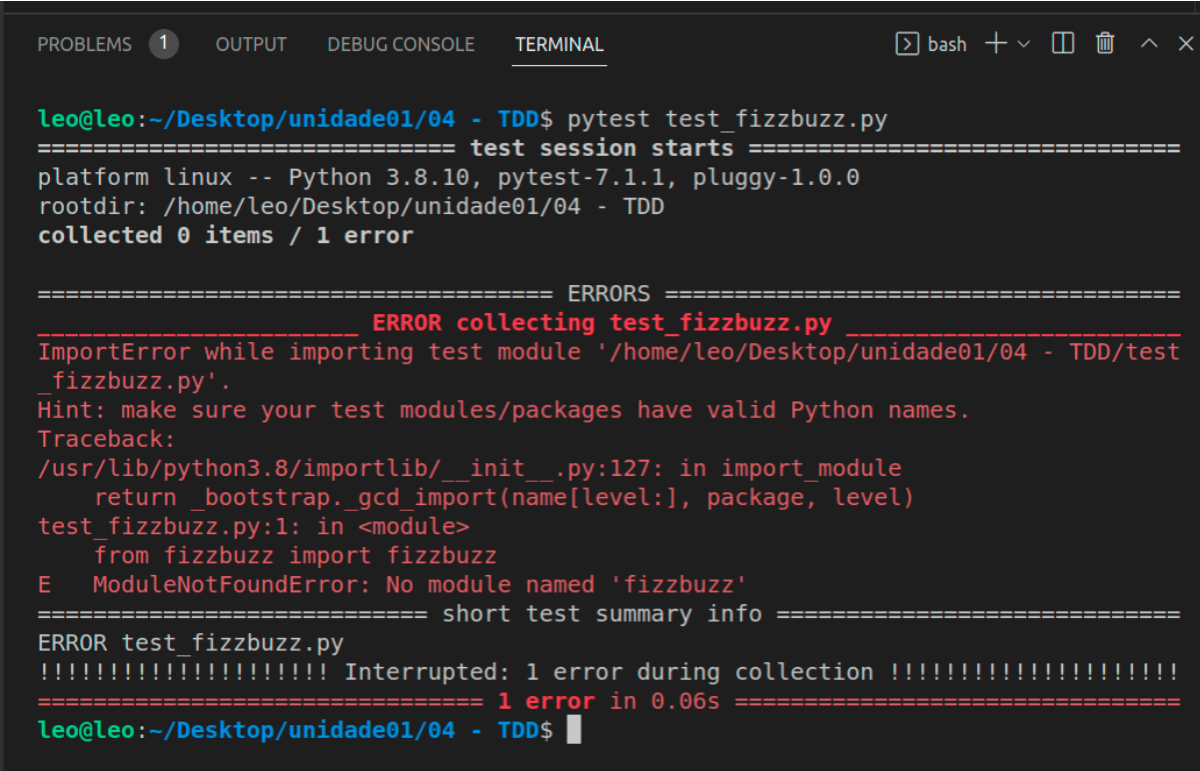
```
from fizzbuzz import fizzbuzz
def test_entrada_1_deve_retornar_1():
```

```
    assert fizzbuzz(1) == "1"
```

Fonte: do autor, 2022.

Seguindo as regras do TDD, o arquivo test_fizz.py foi criado antes do código fizzbuzz.py. Ao executar o pytest, naturalmente ocorrerá erro na execução do programa como apresenta a Figura 4.2.

Figura 4.2. Teste com falha, o arquivo fizzbuzz não existe



```
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL
leo@leo:~/Desktop/unidade01/04 - TDD$ pytest test_fizzbuzz.py
===== test session starts =====
platform linux -- Python 3.8.10, pytest-7.1.1, pluggy-1.0.0
rootdir: /home/leo/Desktop/unidade01/04 - TDD
collected 0 items / 1 error

===== ERRORS =====
_____ ERROR collecting test_fizzbuzz.py _____
ImportError while importing test module '/home/leo/Desktop/unidade01/04 - TDD/test_fizzbuzz.py'.
Hint: make sure your test modules/packages have valid Python names.
Traceback:
/usr/lib/python3.8/importlib/__init__.py:127: in import_module
    return _bootstrap.gcd_import(name[level:], package, level)
test_fizzbuzz.py:1: in <module>
    from fizzbuzz import fizzbuzz
E   ModuleNotFoundError: No module named 'fizzbuzz'
===== short test summary info =====
ERROR test_fizzbuzz.py
!!!!!!!!!!!!!!!!!!!! Interrupted: 1 error during collection !!!!!!!!!!!!!!!!!!!!!
===== 1 error in 0.06s =====
leo@leo:~/Desktop/unidade01/04 - TDD$
```

Fonte: do autor, 2022.

Agora que o teste falhou, o TDD indica para que o programador resolva o problema e faça o teste passar, que neste caso, basta criar o arquivo fizzbuzz com uma função fizzbuzz que receba como parâmetro de entrada um valor inteiro 1 e devolva uma string "1". Para isso, escreva o arquivo fizzbuzz.py conforme apresenta a Codificação 4.2.

Observe que pela Codificação 4.2, o arquivo contém a função fizzbuzz que recebe um valor n, neste momento as validações de tipos serão ignoradas, e devolve uma string. Nenhuma lógica adicional foi incluída. A

função apenas devolve o que o teste espera, que é uma string contendo o valor "1". Ao executar novamente, o teste vai passar, e o pytest vai indicar que a função está correta, conforme "1" apresenta a Figura 4.3.

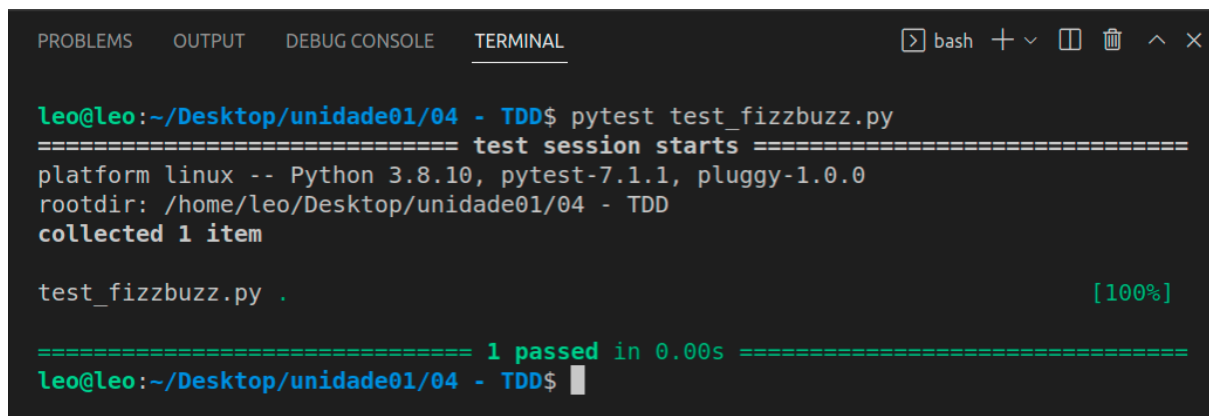
Codificação 4.2. fizzbuzz.py

```
def fizzbuzz(n):
```

```
    return "1"
```

Fonte: do autor, 2022.

Figura 4.3. Primeiro teste passou



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
leo@leo:~/Desktop/unidade01/04 - TDD$ pytest test_fizzbuzz.py
===== test session starts =====
platform linux -- Python 3.8.10, pytest-7.1.1, pluggy-1.0.0
rootdir: /home/leo/Desktop/unidade01/04 - TDD
collected 1 item

test_fizzbuzz.py . [100%]

===== 1 passed in 0.00s =====
leo@leo:~/Desktop/unidade01/04 - TDD$
```

Fonte: do autor, 2022.

Apesar do plano de teste criado, e o programador conhecer antecipadamente as entradas e saídas esperadas, o TDD sugere realizar os testes de unidade em pequenos incrementos que ele nomeia como “*baby steps*”. A construção da funcionalidade ocorre a cada passo e o programador pode pensar em cada entrada sobre como a função deve se comportar. Agora, o próximo passo é tratar o caso de teste CT0002, e para isso inclua o teste conforme a Codificação 4.3.

Codificação 4.3. Tratando o caso de teste CT0002

```
from fizzbuzz import fizzbuzz
```

```
def test_entrada_1_deve_retornar_1():
```

```
    assert fizzbuzz(1) == "1"
```

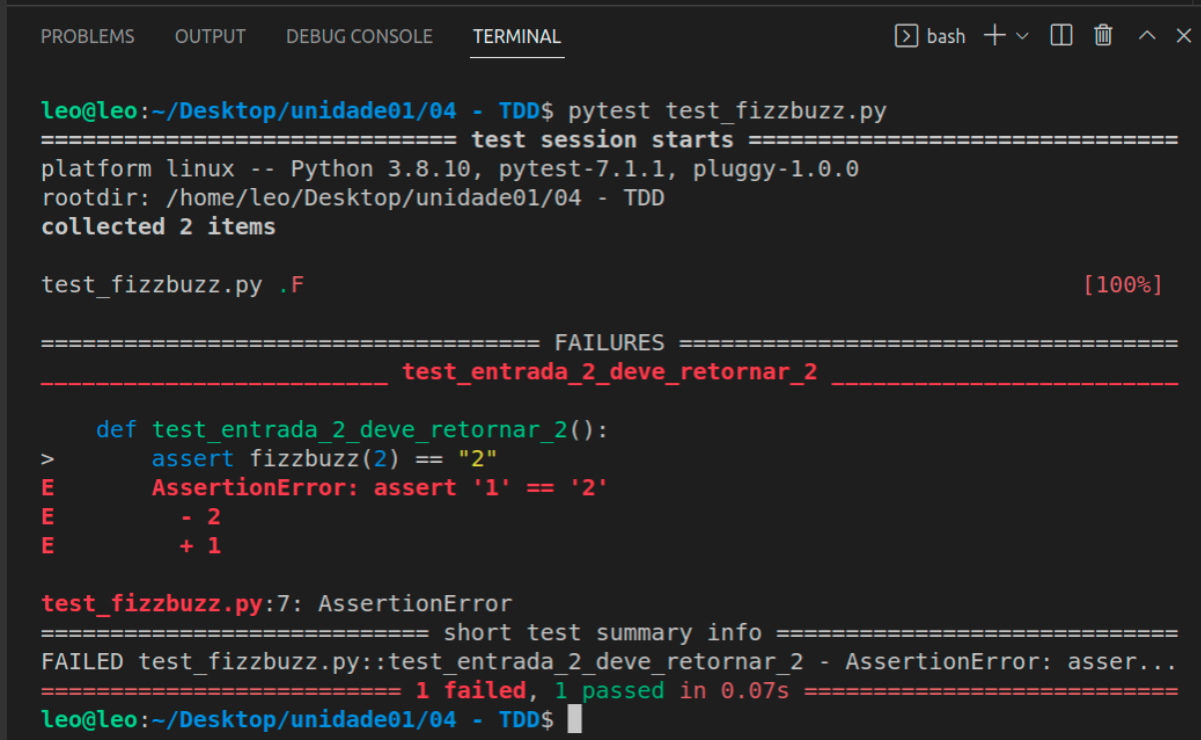
```
def test_entrada_2_deve_retornar_2():
```

```
    assert fizzbuzz(2) == "2"
```

Fonte: do autor, 2022.

Como visto, a função `fizzbuzz`, até o momento, devolve a string "1", e ao executar o `pytest`, certamente a função de teste `test_entrada_2_deve_retornar_2()` deve falhar, conforme apresenta a Figura 4.4.

Figura 4.4. Falha ao executar o teste que trata CT0002



```
leo@leo:~/Desktop/unidade01/04 - TDD$ pytest test_fizzbuzz.py
===== test session starts =====
platform linux -- Python 3.8.10, pytest-7.1.1, pluggy-1.0.0
rootdir: /home/leo/Desktop/unidade01/04 - TDD
collected 2 items

test_fizzbuzz.py .F [100%]

===== FAILURES =====
_____ test_entrada_2_deve_retornar_2 _____

    def test_entrada_2_deve_retornar_2():
>     assert fizzbuzz(2) == "2"
E       AssertionError: assert '1' == '2'
E         - 2
E         + 1

test_fizzbuzz.py:7: AssertionError
===== short test summary info =====
FAILED test_fizzbuzz.py::test_entrada_2_deve_retornar_2 - AssertionError: asser...
===== 1 failed, 1 passed in 0.07s =====
leo@leo:~/Desktop/unidade01/04 - TDD$
```

Fonte: do autor, 2022.

Para corrigir esse teste, a função "fizzbuzz", ao receber um valor 2, deve devolver "2", conforme apresenta a Codificação 4.4.

Codificação 4.4. Para o valor 2 devolva "2"

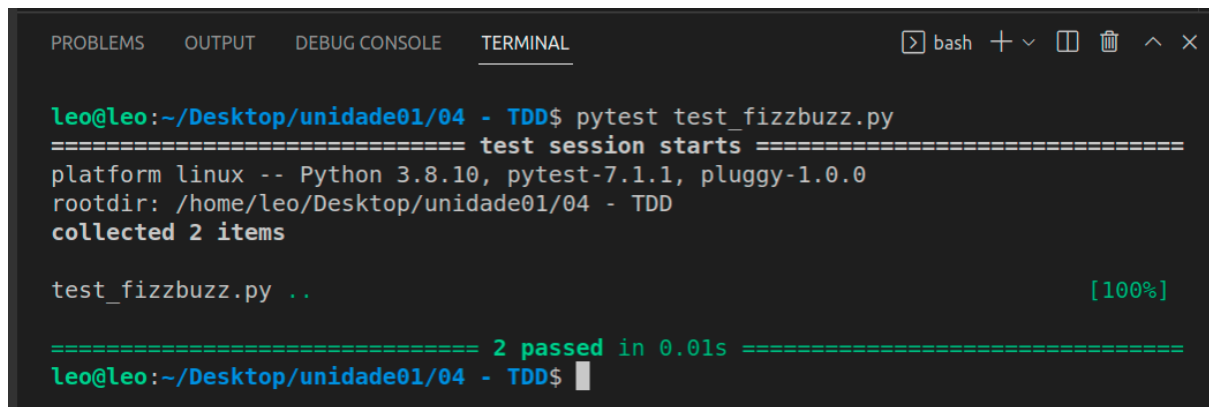
```
def fizzbuzz(n):
    if n == 2:
        return "2"

    return "1"
```

Fonte: do autor, 2022.

Após a alteração, execute o `pytest` e verifique que o teste passou e o caso de teste CT0002 está concluído, conforme a Figura 4.5.

Figura 4.5. Teste para tratar o caso de teste CT0002 passou

A terminal window with a dark background and light-colored text. The window has tabs at the top: 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', and 'TERMINAL'. The 'TERMINAL' tab is active. The prompt is 'leo@leo:~/Desktop/unidade01/04 - TDD\$'. The command 'pytest test_fizzbuzz.py' has been executed. The output shows 'test session starts' followed by platform and version information, 'collected 2 items', and a progress bar indicating 100% completion. The final result is '2 passed in 0.01s'.

```
leo@leo:~/Desktop/unidade01/04 - TDD$ pytest test_fizzbuzz.py
===== test session starts =====
platform linux -- Python 3.8.10, pytest-7.1.1, pluggy-1.0.0
rootdir: /home/leo/Desktop/unidade01/04 - TDD
collected 2 items

test_fizzbuzz.py ..                                     [100%]

===== 2 passed in 0.01s =====
leo@leo:~/Desktop/unidade01/04 - TDD$
```

Fonte: do autor, 2022.

Para tratar o caso de teste CT0003, escreva o código conforme a Codificação 4.5.

Codificação 4.5. Tratando o caso de teste CT0003

```
from fizzbuzz import fizzbuzz
def test_entrada_1_deve_retornar_1():
    assert fizzbuzz(1) == "1"
def test_entrada_2_deve_retornar_2():
    assert fizzbuzz(2) == "2"
def test_entrada_3_deve_retornar_Fizz():

    assert fizzbuzz(3) == "Fizz"
```

Fonte: do autor, 2022.

Ao executar o pytest indicará uma falha, conforme a Figura 4.6.

Figura 4.6. Erro ao tratar caso de teste CT0003

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
leo@leo:~/Desktop/unidade01/04 - TDD$ pytest test_fizzbuzz.py
===== test session starts =====
platform linux -- Python 3.8.10, pytest-7.1.1, pluggy-1.0.0
rootdir: /home/leo/Desktop/unidade01/04 - TDD
collected 3 items

test_fizzbuzz.py ..F [100%]

===== FAILURES =====
----- test_entrada_3_deve_retornar_Fizz -----
>
def test_entrada_3_deve_retornar_Fizz():
>     assert fizzbuzz(3) == "Fizz"
E       AssertionError: assert '1' == 'Fizz'
E       - Fizz
E       + 1

test_fizzbuzz.py:10: AssertionError
===== short test summary info =====
FAILED test_fizzbuzz.py::test_entrada_3_deve_retornar_Fizz - AssertionError: as...
===== 1 failed, 2 passed in 0.07s =====
leo@leo:~/Desktop/unidade01/04 - TDD$
```

Fonte: do autor, 2022.

Até o momento, a função "fizzbuzz" devolve a string "1" ou a string "2". Agora, a função deve tratar caso o valor de n seja igual a 3 e devolver a string "Fizz", conforme a Codificação 4.6.

Codificação 4.6. Para o valor 3 devolva "Fizz"

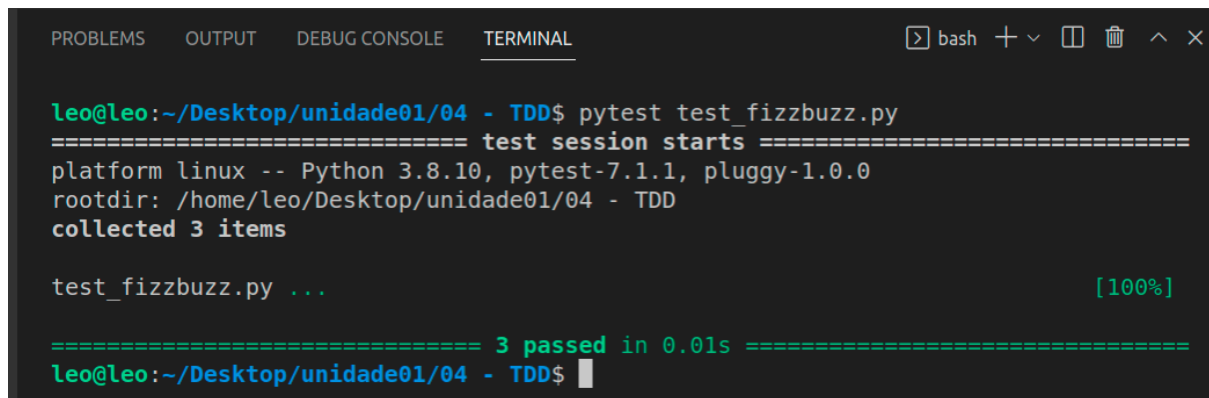
```
def fizzbuzz(n):
    if n == 3:
        return "Fizz"
    if n == 2:
        return "2"

    return "1"
```

Fonte: do autor, 2022.

Para executar os testes de unidade, execute a instrução pytest, ou pytest seguido do nome do arquivo de teste, conforme apresenta a Figura 4.7.

Figura 4.7. A função para tratar CT0003 passou

A terminal window with tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, and TERMINAL. The TERMINAL tab is active, showing a bash prompt. The user runs 'pytest test_fizzbuzz.py'. The output shows 'test session starts', platform and version information, 'collected 3 items', and 'test_fizzbuzz.py ... [100%]'. It concludes with '3 passed in 0.01s' and returns to the prompt.

```
leo@leo:~/Desktop/unidade01/04 - TDD$ pytest test_fizzbuzz.py
===== test session starts =====
platform linux -- Python 3.8.10, pytest-7.1.1, pluggy-1.0.0
rootdir: /home/leo/Desktop/unidade01/04 - TDD
collected 3 items

test_fizzbuzz.py ... [100%]

===== 3 passed in 0.01s =====
leo@leo:~/Desktop/unidade01/04 - TDD$
```

Fonte: do autor, 2022.

Perceba, que o Pytest executou 3 funções de teste que passaram com sucesso. Para cada função que executou com sucesso, o relatório do Pytest apresenta um ponto (.) na cor verde.

Para o caso de teste CT0004, deve-se criar a função que forneça o número 4 para a função "fizzbuzz" e devolva a string "4", conforme a Codificação 4.7.

Codificação 4.7. Tratando o caso de teste CT0004

```
from fizzbuzz import fizzbuzz
def test_entrada_1_deve_retornar_1():
    assert fizzbuzz(1) == "1"
def test_entrada_2_deve_retornar_2():
    assert fizzbuzz(2) == "2"
def test_entrada_3_deve_retornar_Fizz():
    assert fizzbuzz(3) == "Fizz"
def test_entrada_4_deve_retornar_4():
    assert fizzbuzz(4) == "4"
```

Fonte: do autor, 2022.

Ao executar o pytest, observe que, novamente, o teste vai falhar gerando um AssertionError, conforme a Figura 4.8.

Figura 4.8. Falha ao tratar caso de teste CT0004

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
leo@leo:~/Desktop/unidade01/04 - TDD$ pytest test_fizzbuzz.py
===== test session starts =====
platform linux -- Python 3.8.10, pytest-7.1.1, pluggy-1.0.0
rootdir: /home/leo/Desktop/unidade01/04 - TDD
collected 4 items

test_fizzbuzz.py ...F [100%]

===== FAILURES =====
----- test_entrada_4_deve_retornar_4 -----

    def test_entrada_4_deve_retornar_4():
>     assert fizzbuzz(4) == "4"
E       AssertionError: assert '1' == '4'
E         - 4
E         + 1

test_fizzbuzz.py:13: AssertionError
===== short test summary info =====
FAILED test_fizzbuzz.py::test_entrada_4_deve_retornar_4 - AssertionError: asser...
===== 1 failed, 3 passed in 0.07s =====
leo@leo:~/Desktop/unidade01/04 - TDD$
```

Fonte: do autor, 2022.

Altere a função "fizzbuzz" para devolver a string "4" caso o valor de n seja 4, conforme a Codificação 4.8.

Codificação 4.8. Para o valor 4 devolva "4"

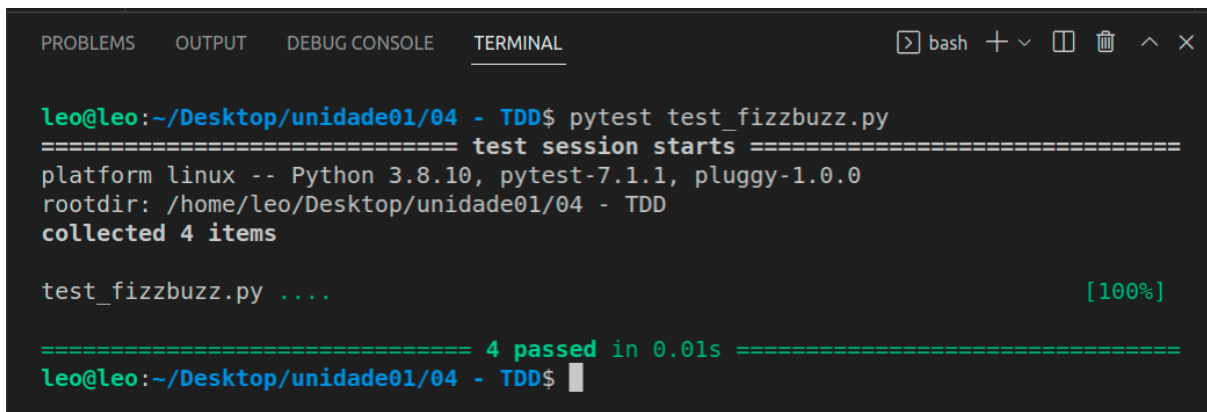
```
def fizzbuzz(n):
    if n == 4:
        return "4"
    if n == 3:
        return "Fizz"
    if n == 2:
        return "2"

    return "1"
```

Fonte: do autor, 2022.

Agora, execute o pytest para verificar que a função "fizzbuzz" está executando corretamente, conforme os casos de testes, conforme a Figura 4.9.

Figura 4.9. Função para tratar o caso de teste CT0004, está passando

A terminal window with a dark background and light-colored text. The window has tabs at the top: 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', and 'TERMINAL'. The 'TERMINAL' tab is active. The prompt is 'leo@leo:~/Desktop/unidade01/04 - TDD\$'. The command 'pytest test_fizzbuzz.py' has been executed. The output shows the test session starting, the platform and Python version (Linux, Python 3.8.10, pytest-7.1.1, pluggy-1.0.0), the root directory, and that 4 items were collected. The test results show 4 passed tests in 0.01s, with a progress bar at 100%.

```
leo@leo:~/Desktop/unidade01/04 - TDD$ pytest test_fizzbuzz.py
===== test session starts =====
platform linux -- Python 3.8.10, pytest-7.1.1, pluggy-1.0.0
rootdir: /home/leo/Desktop/unidade01/04 - TDD
collected 4 items

test_fizzbuzz.py .... [100%]

===== 4 passed in 0.01s =====
leo@leo:~/Desktop/unidade01/04 - TDD$
```

Fonte: do autor, 2022.

Perceba que a função fizzbuzz contém uma série de instruções condicionais (instruções ifs). Perceba que há um padrão, que é para entradas com números diferentes do valor 3, devolva a string com o próprio número. Agora que os testes estão passando, é possível refatorar, de maneira que continue executando corretamente, conforme a Codificação 4.9.

Codificação 4.9. Refatorando a função fizzbuzz

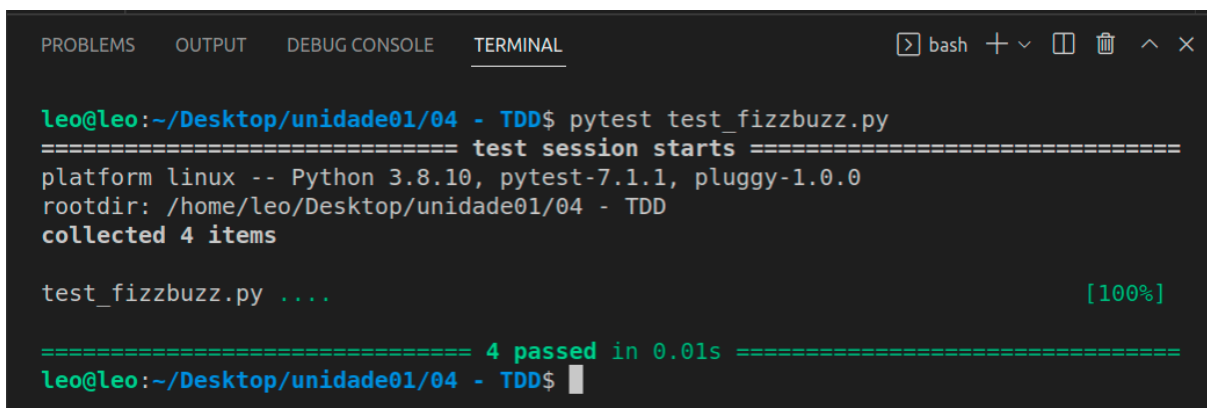
```
def fizzbuzz(n):
    if n == 3:
        return "Fizz"

    return str(n)
```

Fonte: do autor, 2022.

Execute o pytest novamente e verifique que o código ainda continua passando nos testes, conforme apresenta a Figura 4.10.

Figura 4.10. Função para tratar o caso de teste CT0004, está passando após refatoração

A terminal window identical to the one above, showing the same pytest command and output. The test results are the same: 4 passed in 0.01s, with a progress bar at 100%.

```
leo@leo:~/Desktop/unidade01/04 - TDD$ pytest test_fizzbuzz.py
===== test session starts =====
platform linux -- Python 3.8.10, pytest-7.1.1, pluggy-1.0.0
rootdir: /home/leo/Desktop/unidade01/04 - TDD
collected 4 items

test_fizzbuzz.py .... [100%]

===== 4 passed in 0.01s =====
leo@leo:~/Desktop/unidade01/04 - TDD$
```

Fonte: do autor, 2022.

O caso de teste CT0005, espera que o parâmetro de entrada de fizzbuzz seja um número 5, e devolva a string "Buzz". Inclua o teste conforme a Figura 4.10.

Codificação 4.10. Tratando o caso de teste CT0005

```
from fizzbuzz import fizzbuzz
def test_entrada_1_deve_retornar_1():
    assert fizzbuzz(1) == "1"
def test_entrada_2_deve_retornar_2():
    assert fizzbuzz(2) == "2"
def test_entrada_3_deve_retornar_Fizz():
    assert fizzbuzz(3) == "Fizz"
def test_entrada_4_deve_retornar_4():
    assert fizzbuzz(4) == "4"
def test_entrada_5_deve_retornar_Buzz():

    assert fizzbuzz(5) == "Buzz"
```

Fonte: do autor, 2022.

Execute o pytest e verifique que o teste para o CT0005 irá falhar, conforme a Figura 4.11.

Figura 4.11. Função para tratar CT0005 com falhas

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
leo@leo:~/Desktop/unidade01/04 - TDD$ pytest test_fizzbuzz.py
===== test session starts =====
platform linux -- Python 3.8.10, pytest-7.1.1, pluggy-1.0.0
rootdir: /home/leo/Desktop/unidade01/04 - TDD
collected 5 items

test_fizzbuzz.py ....F [100%]

===== FAILURES =====
_____ test_entrada_5_deve_retornar_Buzz _____

    def test_entrada_5_deve_retornar_Buzz():
>     assert fizzbuzz(5) == "Buzz"
E       AssertionError: assert '5' == 'Buzz'
E         - Buzz
E         + 5

test_fizzbuzz.py:16: AssertionError
===== short test summary info =====
FAILED test_fizzbuzz.py::test_entrada_5_deve_retornar_Buzz - AssertionError: as...
===== 1 failed, 4 passed in 0.07s =====
leo@leo:~/Desktop/unidade01/04 - TDD$
```

Fonte: do autor, 2022.

Altere o código da função fizzbuzz, para o tratar o caso de teste CT0005, conforme a Codificação 4.11.

Codificação 4.11. Tratando o caso de teste CT0005

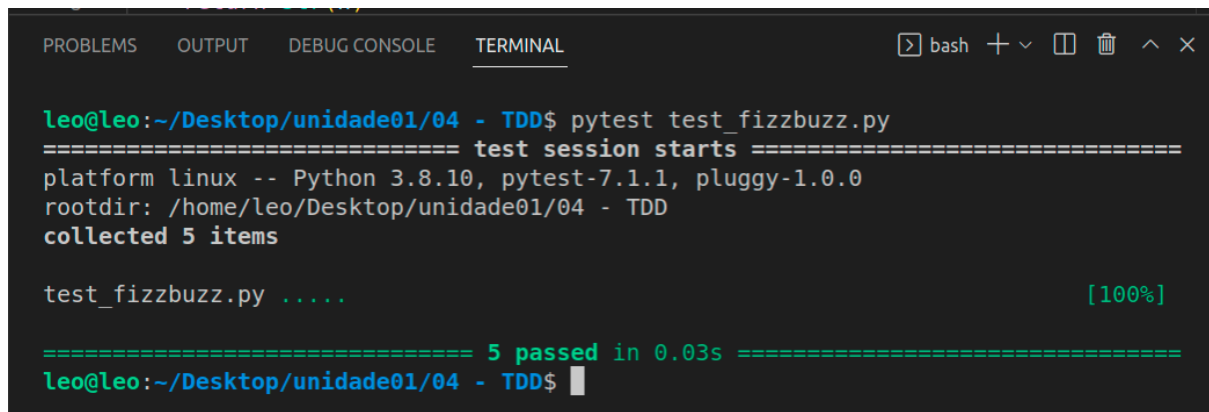
```
def fizzbuzz(n):
    if n == 5:
        return "Buzz"
    if n == 3:
        return "Fizz"

    return str(n)
```

Fonte: do autor, 2022.

Agora, execute o pytest e verifique que o caso de teste CT0005 está contemplado pela função fizzbuzz, conforme a Figura 4.12.

Figura 4.12. Teste para o caso de teste CT0005 passando



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
leo@leo:~/Desktop/unidade01/04 - TDD$ pytest test_fizzbuzz.py
===== test session starts =====
platform linux -- Python 3.8.10, pytest-7.1.1, pluggy-1.0.0
rootdir: /home/leo/Desktop/unidade01/04 - TDD
collected 5 items

test_fizzbuzz.py ..... [100%]

===== 5 passed in 0.03s =====
leo@leo:~/Desktop/unidade01/04 - TDD$
```

Fonte: do autor, 2022.

Para o caso de testes CT0006, dado o valor de entrada $n=6$, a função `fizzbuzz` deve retornar "Fizz", pois n é divisível por 3, conforme as condições descritas para o problema. Para isso, inclua o teste conforme a Codificação 4.12.

Codificação 4.12. Tratando o caso de teste CT0006

```
from fizzbuzz import fizzbuzz
def test_entrada_1_deve_retornar_1():
    assert fizzbuzz(1) == "1"
def test_entrada_2_deve_retornar_2():
    assert fizzbuzz(2) == "2"
def test_entrada_3_deve_retornar_Fizz():
    assert fizzbuzz(3) == "Fizz"
def test_entrada_4_deve_retornar_4():
    assert fizzbuzz(4) == "4"
def test_entrada_5_deve_retornar_Buzz():
    assert fizzbuzz(5) == "Buzz"
def test_entrada_6_deve_retornar_Fizz():
    assert fizzbuzz(6) == "Fizz"
```

Fonte: do autor, 2022.

Execute o `pytest` e verifique que o CT0006 ainda não está contemplado pela função `"fizzbuzz"`, conforme a Figura 4.13.

Figura 4.13. Falha ao testar função que trata CT0006


```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
leo@leo:~/Desktop/unidade01/04 - TDD$ pytest test fizzbuzz.py
===== test session starts =====
platform linux -- Python 3.8.10, pytest-7.1.1, pluggy-1.0.0
rootdir: /home/leo/Desktop/unidade01/04 - TDD
collected 6 items

test_fizzbuzz.py .....F [100%]

===== FAILURES =====
----- test_entrada_6_deve_retornar_Fizz -----

    def test_entrada_6_deve_retornar_Fizz():
>     assert fizzbuzz(6) == "Fizz"
E       AssertionError: assert '6' == 'Fizz'
E       - Fizz
E       + 6

test_fizzbuzz.py:19: AssertionError
===== short test summary info =====
FAILED test_fizzbuzz.py::test_entrada_6_deve_retornar_Fizz - AssertionError: as...
===== 1 failed, 5 passed in 0.10s =====
leo@leo:~/Desktop/unidade01/04 - TDD$
```

Fonte: do autor, 2022.

Para tratar o caso de teste CT0006, pode-se verificar se dado n como entrada, n seja divisível por 3. Para isso, basta verificar se o resto da divisão de n por 3 é igual a 0. Observe a alteração na função `fizzbuzz`, conforme a Codificação 4.13.

Codificação 4.13. Tratando o caso de teste CT0006

```
def fizzbuzz(n):
    if n == 5:
        return "Buzz"
    if n % 3 == 0:
        return "Fizz"

    return str(n)
```

Fonte: do autor, 2022.

Para realizar os testes, execute a instrução `pytest` seguido do nome do arquivo conforme apresenta a Figura 4.14.

Figura 4.14. Verificando a correção da função `fizzbuzz` para o caso de teste CT0006

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
bash + - [ ] [ ] ^ x

leo@leo:~/Desktop/unidade01/04 - TDD$ pytest test fizzbuzz.py
===== test session starts =====
platform linux -- Python 3.8.10, pytest-7.1.1, pluggy-1.0.0
rootdir: /home/leo/Desktop/unidade01/04 - TDD
collected 6 items

test_fizzbuzz.py ..... [100%]

===== 6 passed in 0.01s =====
leo@leo:~/Desktop/unidade01/04 - TDD$
```

Fonte: do autor, 2022.

Para os casos de testes CT0007, CT0008 e CT0009, a função fizzbuzz executa corretamente, basta criar os testes para esses três casos e verificar, conforme apresenta a Codificação 4.14.

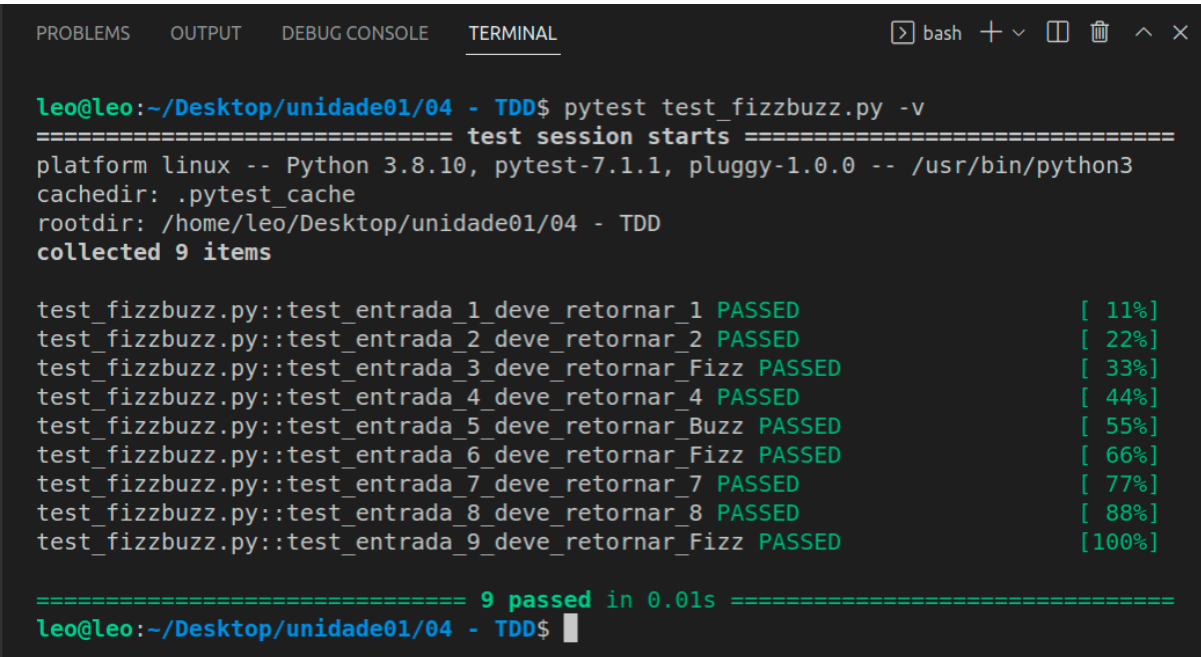
Codificação 4.14. Tratando os casos de teste CT0007, CT0008 e CT0009

```
from fizzbuzz import fizzbuzz
def test_entrada_1_deve_retornar_1():
    assert fizzbuzz(1) == "1"
def test_entrada_2_deve_retornar_2():
    assert fizzbuzz(2) == "2"
def test_entrada_3_deve_retornar_Fizz():
    assert fizzbuzz(3) == "Fizz"
def test_entrada_4_deve_retornar_4():
    assert fizzbuzz(4) == "4"
def test_entrada_5_deve_retornar_Buzz():
    assert fizzbuzz(5) == "Buzz"
def test_entrada_6_deve_retornar_Fizz():
    assert fizzbuzz(6) == "Fizz"
def test_entrada_7_deve_retornar_7():
    assert fizzbuzz(7) == "7"
def test_entrada_8_deve_retornar_8():
    assert fizzbuzz(8) == "8"
def test_entrada_9_deve_retornar_Fizz():
    assert fizzbuzz(9) == "Fizz"
```

Fonte: do autor, 2022.

Verifique, pela Figura 4.15, que os testes executam sem falhas. Agora, utilizou-se a opção -v (verbose) para apresentar mais detalhes sobre as execuções de testes.

Figura 4.15. Função para tratar casos de testes CT0007, CT0008 e CT0009 passaram



```
leo@leo:~/Desktop/unidade01/04 - TDD$ pytest test_fizzbuzz.py -v
===== test session starts =====
platform linux -- Python 3.8.10, pytest-7.1.1, pluggy-1.0.0 -- /usr/bin/python3
cachedir: .pytest_cache
rootdir: /home/leo/Desktop/unidade01/04 - TDD
collected 9 items

test_fizzbuzz.py::test_entrada_1_deve_retornar_1 PASSED [ 11%]
test_fizzbuzz.py::test_entrada_2_deve_retornar_2 PASSED [ 22%]
test_fizzbuzz.py::test_entrada_3_deve_retornar_Fizz PASSED [ 33%]
test_fizzbuzz.py::test_entrada_4_deve_retornar_4 PASSED [ 44%]
test_fizzbuzz.py::test_entrada_5_deve_retornar_Buzz PASSED [ 55%]
test_fizzbuzz.py::test_entrada_6_deve_retornar_Fizz PASSED [ 66%]
test_fizzbuzz.py::test_entrada_7_deve_retornar_7 PASSED [ 77%]
test_fizzbuzz.py::test_entrada_8_deve_retornar_8 PASSED [ 88%]
test_fizzbuzz.py::test_entrada_9_deve_retornar_Fizz PASSED [100%]

===== 9 passed in 0.01s =====
leo@leo:~/Desktop/unidade01/04 - TDD$
```

Fonte: do autor, 2022.

Para o caso de teste CT0010, dado o valor de entrada n=10, a saída esperada é a palavra "Buzz", pois 10 é divisível por 5. Basta criar a função para tratar o caso de teste CT0010, conforme a Codificação 4.16.

Codificação 4.16. Tratando o caso de teste CT0010

```
from fizzbuzz import fizzbuzz

def test_entrada_1_deve_retornar_1():
    assert fizzbuzz(1) == "1"

def test_entrada_2_deve_retornar_2():
    assert fizzbuzz(2) == "2"

def test_entrada_3_deve_retornar_Fizz():
    assert fizzbuzz(3) == "Fizz"

def test_entrada_4_deve_retornar_4():
    assert fizzbuzz(4) == "4"
```

```

def test_entrada_5_deve_retornar_Buzz():
    assert fizzbuzz(5) == "Buzz"
def test_entrada_6_deve_retornar_Fizz():
    assert fizzbuzz(6) == "Fizz"
def test_entrada_7_deve_retornar_7():
    assert fizzbuzz(7) == "7"
def test_entrada_8_deve_retornar_8():
    assert fizzbuzz(8) == "8"
def test_entrada_9_deve_retornar_Fizz():
    assert fizzbuzz(9) == "Fizz"
def test_entrada_10_deve_retornar_Buzz():

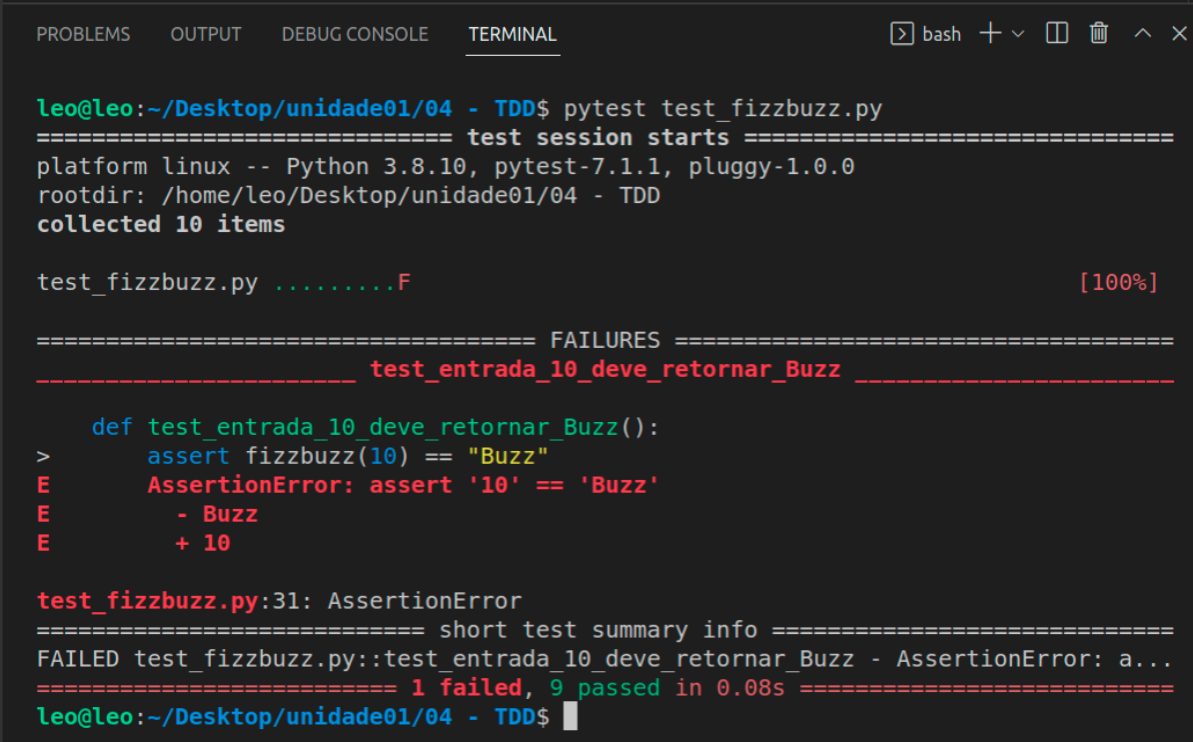
    assert fizzbuzz(10) == "Buzz"

```

Fonte: do autor, 2022.

O caso de teste CT0010 ainda não está tratado pela função "fizzbuzz", conforme indica o relatório do pytest na Figura 4.16.

Figura 4.16. Falha ao tratar CT0010



```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
leo@leo:~/Desktop/unidade01/04 - TDD$ pytest test_fizzbuzz.py
===== test session starts =====
platform linux -- Python 3.8.10, pytest-7.1.1, pluggy-1.0.0
rootdir: /home/leo/Desktop/unidade01/04 - TDD
collected 10 items

test_fizzbuzz.py .....F [100%]

===== FAILURES =====
----- test_entrada_10_deve_retornar_Buzz -----
>
def test_entrada_10_deve_retornar_Buzz():
>     assert fizzbuzz(10) == "Buzz"
E       AssertionError: assert '10' == 'Buzz'
E       - Buzz
E       + 10

test_fizzbuzz.py:31: AssertionError
===== short test summary info =====
FAILED test_fizzbuzz.py::test_entrada_10_deve_retornar_Buzz - AssertionError: a...
===== 1 failed, 9 passed in 0.08s =====
leo@leo:~/Desktop/unidade01/04 - TDD$

```

Fonte: do autor, 2022.

Para tratar o caso de teste CT0010, basta verificar para dado n de entrada, se n é divisível por 5, caso afirmativo, a função deve devolver "Buzz", conforme as condições impostas pelo problema FizzBuzz. O código para tratar esta alteração está em Codificação 4.17.

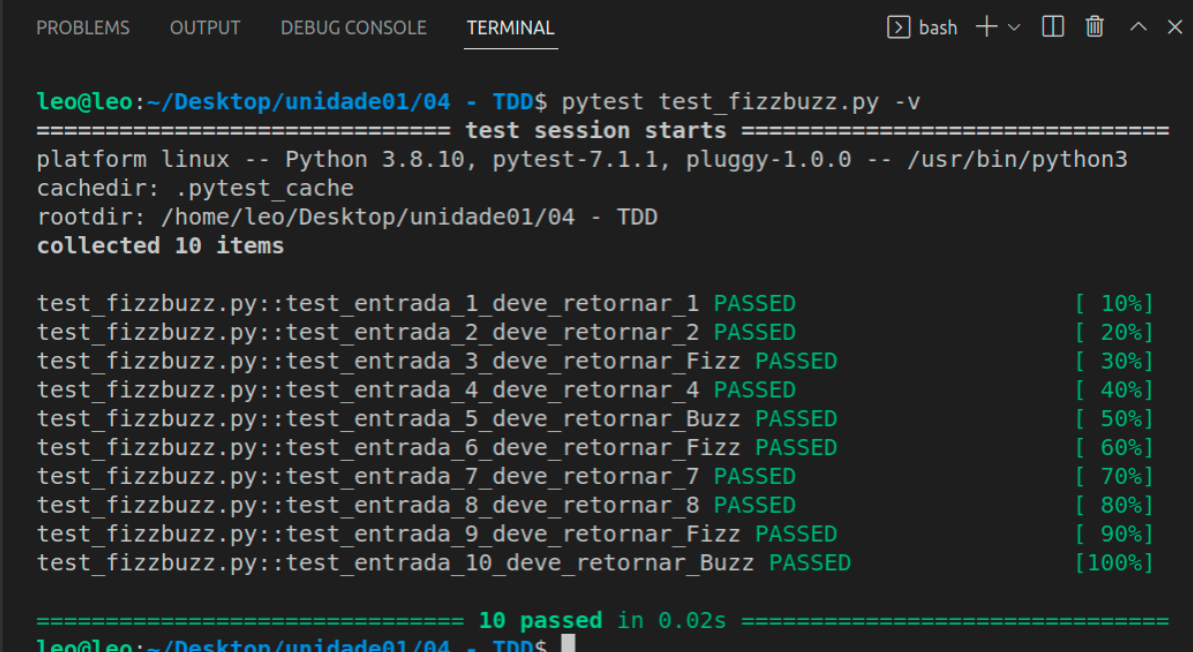
Codificação 4.17. Tratando o caso de teste CT0010

```
def fizzbuzz(n):  
    if n % 5 == 0:  
        return "Buzz"  
    if n % 3 == 0:  
        return "Fizz"  
  
    return str(n)
```

Fonte: do autor, 2022.

Após alteração da função fizzbuzz, execute o pytest novamente e verifique que o teste para tratar o caso de teste CT0010 passou, conforme a Figura 4.17.

Figura 4.17. Verificando as funções de tratamento dos casos de testes



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
leo@leo:~/Desktop/unidade01/04 - TDD$ pytest test_fizzbuzz.py -v
===== test session starts =====
platform linux -- Python 3.8.10, pytest-7.1.1, pluggy-1.0.0 -- /usr/bin/python3
cachedir: .pytest_cache
rootdir: /home/leo/Desktop/unidade01/04 - TDD
collected 10 items

test_fizzbuzz.py::test_entrada_1_deve_retornar_1 PASSED [ 10%]
test_fizzbuzz.py::test_entrada_2_deve_retornar_2 PASSED [ 20%]
test_fizzbuzz.py::test_entrada_3_deve_retornar_Fizz PASSED [ 30%]
test_fizzbuzz.py::test_entrada_4_deve_retornar_4 PASSED [ 40%]
test_fizzbuzz.py::test_entrada_5_deve_retornar_Buzz PASSED [ 50%]
test_fizzbuzz.py::test_entrada_6_deve_retornar_Fizz PASSED [ 60%]
test_fizzbuzz.py::test_entrada_7_deve_retornar_7 PASSED [ 70%]
test_fizzbuzz.py::test_entrada_8_deve_retornar_8 PASSED [ 80%]
test_fizzbuzz.py::test_entrada_9_deve_retornar_Fizz PASSED [ 90%]
test_fizzbuzz.py::test_entrada_10_deve_retornar_Buzz PASSED [100%]

===== 10 passed in 0.02s =====
leo@leo:~/Desktop/unidade01/04 - TDD$
```

Fonte: do autor, 2022.

Com isso, finalizaram-se os casos de testes que foram planejados no início desta parte, conforme a Tabela 4.1. Mas ainda é necessário tratar o caso em

que dado n de entrada para a função `fizzbuzz`, caso n seja divisível por 3 e também divisível por 5, a função deve devolver a string "FizzBuzz". Segue, pela Tabela 4.2, mais uma bateria de testes planejados para atender esta demanda.

Tabela 4.2. Planejando os casos de testes adicionais

<u>fizzbuzz(n)</u>		
#caso de teste	Parâmetros de entrada	Resultado esperado
CT0001	$n=1$	<u>1</u>
CT0002	$n=2$	<u>2</u>
CT0003	$n=3$	<u>Fizz</u>
CT0004	$n=4$	<u>4</u>
CT0005	$n=5$	<u>Buzz</u>
CT0006	$n=6$	<u>Fizz</u>
CT0007	$n=7$	<u>7</u>
CT0008	$n=8$	<u>8</u>
CT0009	$n=9$	<u>Fizz</u>
CT0010	$n=10$	<u>Buzz</u>

Fonte: do autor, 2022.

Iniciando pelos casos de testes CT0011, CT0012, CT0013 e CT0014. Todos estes testes são tratados pela função "fizzbuzz", basta incluir as funções para tratamento de testes para verificar a execução correta da função `fizzbuzz`, conforme Codificação 4.18.

Codificação 4.18. Tratando o caso de teste CT001006

```
from fizzbuzz import fizzbuzz
```

```
# funções para tratamentos dos casos de teste de CT0001 até CT0010.
```

```

def test_entrada_11_deve_retornar_11():
    assert fizzbuzz(11) == "11"
def test_entrada_12_deve_retornar_Fizz():
    assert fizzbuzz(12) == "Fizz"
def test_entrada_13_deve_retornar_13():
    assert fizzbuzz(13) == "13"
def test_entrada_14_deve_retornar_14():

    assert fizzbuzz(14) == "14"

```

Fonte: do autor, 2022.

Ao executar o pytest, pode-se verificar pelo relatório que as funções de testes executaram corretamente, observe a Figura 4.18.

Figura 4.18. Executando as funções de teste pelo pytest

```

leo@leo:~/Desktop/unidade01/04 - TDD$ pytest test_fizzbuzz.py -v
===== test session starts =====
platform linux -- Python 3.8.10, pytest-7.1.1, pluggy-1.0.0 -- /usr/bin/python3
cachedir: .pytest_cache
rootdir: /home/leo/Desktop/unidade01/04 - TDD
collected 14 items

test_fizzbuzz.py::test_entrada_1_deve_retornar_1 PASSED [ 7%]
test_fizzbuzz.py::test_entrada_2_deve_retornar_2 PASSED [ 14%]
test_fizzbuzz.py::test_entrada_3_deve_retornar_Fizz PASSED [ 21%]
test_fizzbuzz.py::test_entrada_4_deve_retornar_4 PASSED [ 28%]
test_fizzbuzz.py::test_entrada_5_deve_retornar_Buzz PASSED [ 35%]
test_fizzbuzz.py::test_entrada_6_deve_retornar_Fizz PASSED [ 42%]
test_fizzbuzz.py::test_entrada_7_deve_retornar_7 PASSED [ 50%]
test_fizzbuzz.py::test_entrada_8_deve_retornar_8 PASSED [ 57%]
test_fizzbuzz.py::test_entrada_9_deve_retornar_Fizz PASSED [ 64%]
test_fizzbuzz.py::test_entrada_10_deve_retornar_Buzz PASSED [ 71%]
test_fizzbuzz.py::test_entrada_11_deve_retornar_11 PASSED [ 78%]
test_fizzbuzz.py::test_entrada_12_deve_retornar_Fizz PASSED [ 85%]
test_fizzbuzz.py::test_entrada_13_deve_retornar_13 PASSED [ 92%]
test_fizzbuzz.py::test_entrada_14_deve_retornar_14 PASSED [100%]

===== 14 passed in 0.02s =====
leo@leo:~/Desktop/unidade01/04 - TDD$

```

Fonte: do autor, 2022.

Para tratar o caso de teste CT0015, é preciso verificar que o valor é tanto divisível por 3, como também, divisível por 5. A função de tratamento deste caso de teste é apresentado pela Codificação 4.19.

Codificação 4.19. Tratando o caso de teste CT001015

```

from fizzbuzz import fizzbuzz
# funções para tratamentos dos casos de teste de CT0001 até CT0010.
def test_entrada_11_deve_retornar_11():
    assert fizzbuzz(11) == "11"
def test_entrada_12_deve_retornar_Fizz():
    assert fizzbuzz(12) == "Fizz"
def test_entrada_13_deve_retornar_13():
    assert fizzbuzz(13) == "13"
def test_entrada_14_deve_retornar_14():
    assert fizzbuzz(14) == "14"
def test_entrada_15_deve_retornar_FizzBuzz():

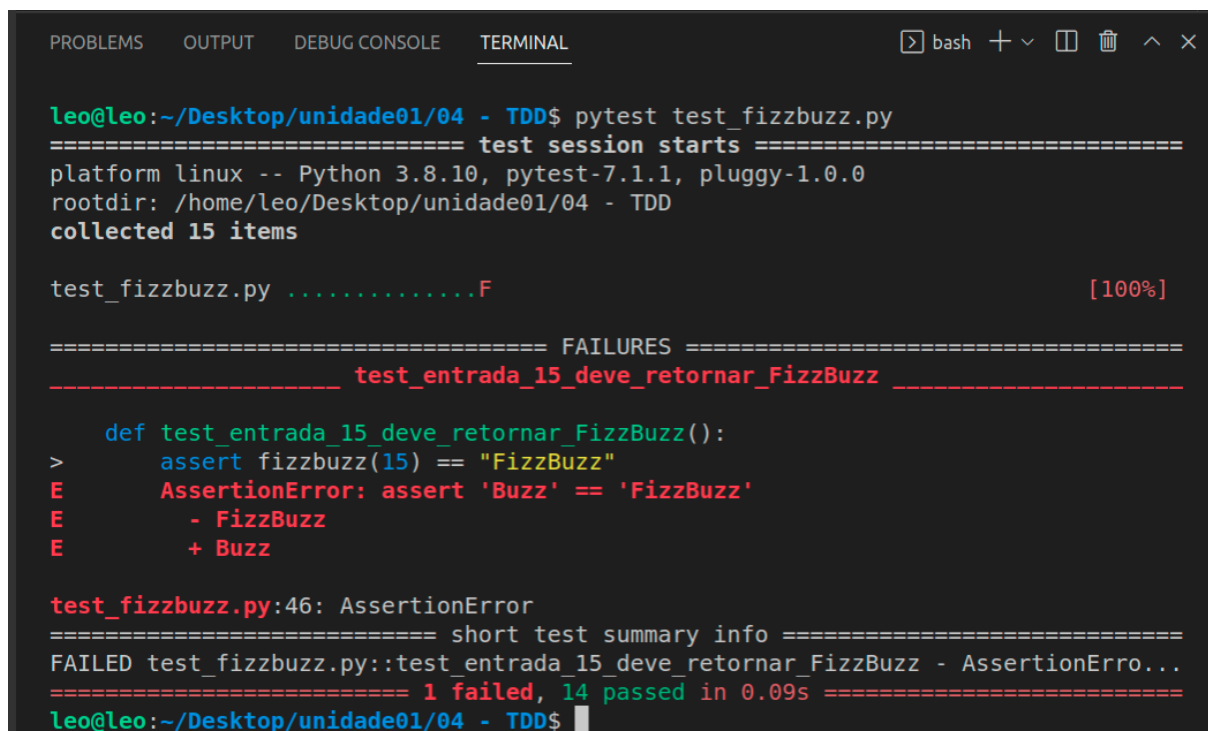
    assert fizzbuzz(15) == "FizzBuzz"

```

Fonte: do autor, 2022.

Ao executar o pytest, a função de tratamento do caso de teste CT0015 irá falhar, conforme apresenta a Figura 4.19.

Figura 4.19. Erro ao tratar caso de teste CT0015



```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
leo@leo:~/Desktop/unidade01/04 - TDD$ pytest test_fizzbuzz.py
===== test session starts =====
platform linux -- Python 3.8.10, pytest-7.1.1, pluggy-1.0.0
rootdir: /home/leo/Desktop/unidade01/04 - TDD
collected 15 items

test_fizzbuzz.py .....F [100%]

===== FAILURES =====
_____ test_entrada_15_deve_retornar_FizzBuzz _____

    def test_entrada_15_deve_retornar_FizzBuzz():
>     assert fizzbuzz(15) == "FizzBuzz"
E       AssertionError: assert 'Buzz' == 'FizzBuzz'
E         - FizzBuzz
E         + Buzz

test_fizzbuzz.py:46: AssertionError
===== short test summary info =====
FAILED test_fizzbuzz.py::test_entrada_15_deve_retornar_FizzBuzz - AssertionErro...
===== 1 failed, 14 passed in 0.09s =====
leo@leo:~/Desktop/unidade01/04 - TDD$

```

Fonte: do autor, 2022.

Para corrigir essa falha, altere o código da função "fizzbuzz", conforme apresenta a Codificação 4.20.

Codificação 4.20. Tratando o caso de teste CT0015

```
def fizzbuzz(n):  
    if n % 3 == 0 and n % 5 == 0:  
        return "FizzBuzz"  
    if n % 5 == 0:  
        return "Buzz"  
    if n % 3 == 0:  
        return "Fizz"  
  
    return str(n)
```

Fonte: do autor, 2022.

Após alterar a função "fizzbuzz", execute o pytest novamente e verifique a função de tratamento do caso de teste executando sem falhas, conforme Figura 4.20.

Figura 4.20. Funções de tratamento de casos de testes executando corretamente

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
leo@leo:~/Desktop/unidade01/04 - TDD$ pytest test_fizzbuzz.py -v
===== test session starts =====
platform linux -- Python 3.8.10, pytest-7.1.1, pluggy-1.0.0 -- /usr/bin/python3
cachedir: .pytest_cache
rootdir: /home/leo/Desktop/unidade01/04 - TDD
collected 15 items

test_fizzbuzz.py::test_entrada_1_deve_retornar_1 PASSED [ 6%]
test_fizzbuzz.py::test_entrada_2_deve_retornar_2 PASSED [ 13%]
test_fizzbuzz.py::test_entrada_3_deve_retornar_Fizz PASSED [ 20%]
test_fizzbuzz.py::test_entrada_4_deve_retornar_4 PASSED [ 26%]
test_fizzbuzz.py::test_entrada_5_deve_retornar_Buzz PASSED [ 33%]
test_fizzbuzz.py::test_entrada_6_deve_retornar_Fizz PASSED [ 40%]
test_fizzbuzz.py::test_entrada_7_deve_retornar_7 PASSED [ 46%]
test_fizzbuzz.py::test_entrada_8_deve_retornar_8 PASSED [ 53%]
test_fizzbuzz.py::test_entrada_9_deve_retornar_Fizz PASSED [ 60%]
test_fizzbuzz.py::test_entrada_10_deve_retornar_Buzz PASSED [ 66%]
test_fizzbuzz.py::test_entrada_11_deve_retornar_11 PASSED [ 73%]
test_fizzbuzz.py::test_entrada_12_deve_retornar_Fizz PASSED [ 80%]
test_fizzbuzz.py::test_entrada_13_deve_retornar_13 PASSED [ 86%]
test_fizzbuzz.py::test_entrada_14_deve_retornar_14 PASSED [ 93%]
test_fizzbuzz.py::test_entrada_15_deve_retornar_FizzBuzz PASSED [100%]

===== 15 passed in 0.02s =====
leo@leo:~/Desktop/unidade01/04 - TDD$
```

Fonte: do autor, 2022.

Para os casos de testes CT0016, CT0017, CT0018, CT0019 e CT0020, a função "fizzbuzz" é executada corretamente, para testar estes casos de testes, inclua os testes conforme a Codificação 4.21.

Codificação 4.21. Tratando o caso de teste CT001015

```
from fizzbuzz import fizzbuzz
# funções para tratamentos dos casos de teste de CT0001 até CT0010.
def test_entrada_11_deve_retornar_11():
    assert fizzbuzz(11) == "11"
def test_entrada_12_deve_retornar_Fizz():
    assert fizzbuzz(12) == "Fizz"
def test_entrada_13_deve_retornar_13():
    assert fizzbuzz(13) == "13"
def test_entrada_14_deve_retornar_14():
    assert fizzbuzz(14) == "14"
def test_entrada_15_deve_retornar_FizzBuzz():
    assert fizzbuzz(15) == "FizzBuzz"
```

```

def test_entrada_16_deve_retornar_16():
    assert fizzbuzz(16) == "16"
def test_entrada_17_deve_retornar_17():
    assert fizzbuzz(17) == "17"
def test_entrada_18_deve_retornar_18():
    assert fizzbuzz(18) == "Fizz"
def test_entrada_19_deve_retornar_19():
    assert fizzbuzz(19) == "19"
def test_entrada_20_deve_retornar_Buzz():

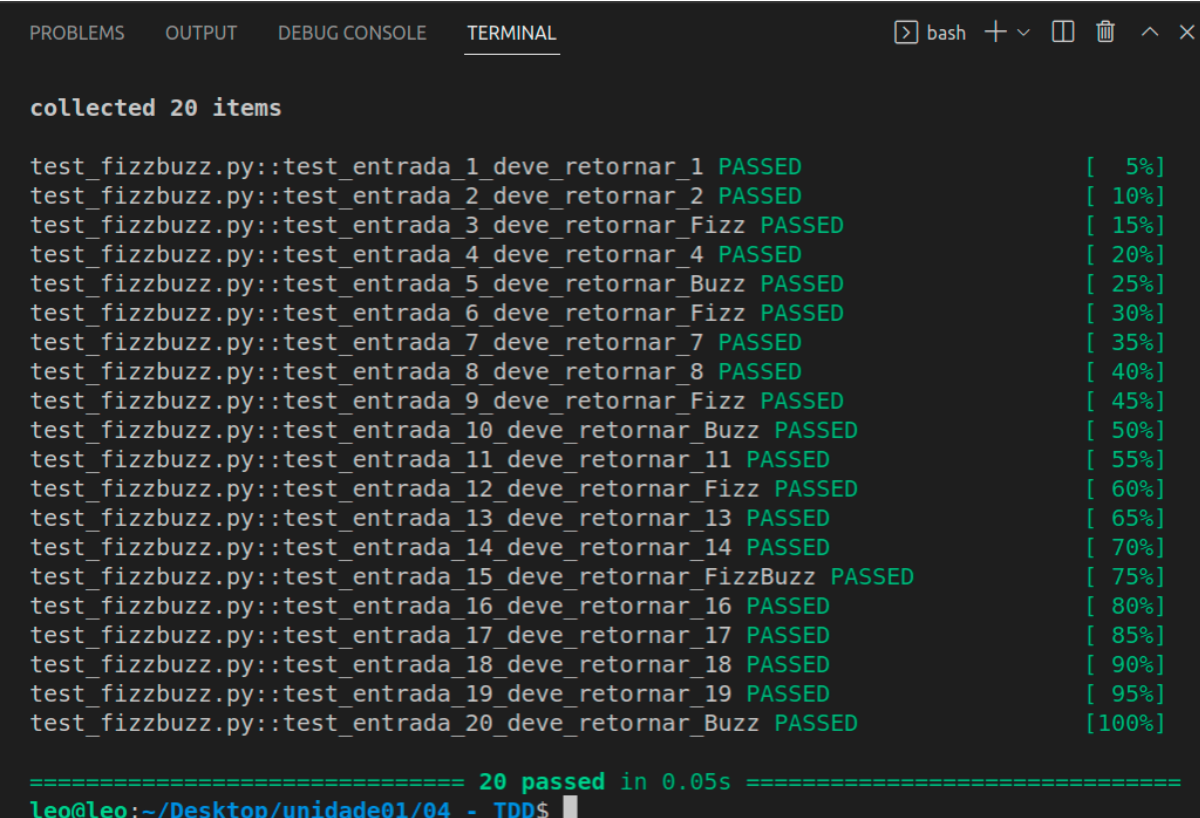
    assert fizzbuzz(20) == "Buzz"

```

Fonte: do autor, 2022.

Ao executar o pytest, as funções de tratamento dos casos de testes executam corretamente, conforme ilustra a Figura 4.21.

Figura 4.21. Funções de testes passando



The screenshot shows a terminal window with the following output:

```

collected 20 items

test_fizzbuzz.py::test_entrada_1_deve_retornar_1 PASSED [ 5%]
test_fizzbuzz.py::test_entrada_2_deve_retornar_2 PASSED [ 10%]
test_fizzbuzz.py::test_entrada_3_deve_retornar_Fizz PASSED [ 15%]
test_fizzbuzz.py::test_entrada_4_deve_retornar_4 PASSED [ 20%]
test_fizzbuzz.py::test_entrada_5_deve_retornar_Buzz PASSED [ 25%]
test_fizzbuzz.py::test_entrada_6_deve_retornar_Fizz PASSED [ 30%]
test_fizzbuzz.py::test_entrada_7_deve_retornar_7 PASSED [ 35%]
test_fizzbuzz.py::test_entrada_8_deve_retornar_8 PASSED [ 40%]
test_fizzbuzz.py::test_entrada_9_deve_retornar_Fizz PASSED [ 45%]
test_fizzbuzz.py::test_entrada_10_deve_retornar_Buzz PASSED [ 50%]
test_fizzbuzz.py::test_entrada_11_deve_retornar_11 PASSED [ 55%]
test_fizzbuzz.py::test_entrada_12_deve_retornar_Fizz PASSED [ 60%]
test_fizzbuzz.py::test_entrada_13_deve_retornar_13 PASSED [ 65%]
test_fizzbuzz.py::test_entrada_14_deve_retornar_14 PASSED [ 70%]
test_fizzbuzz.py::test_entrada_15_deve_retornar_FizzBuzz PASSED [ 75%]
test_fizzbuzz.py::test_entrada_16_deve_retornar_16 PASSED [ 80%]
test_fizzbuzz.py::test_entrada_17_deve_retornar_17 PASSED [ 85%]
test_fizzbuzz.py::test_entrada_18_deve_retornar_18 PASSED [ 90%]
test_fizzbuzz.py::test_entrada_19_deve_retornar_19 PASSED [ 95%]
test_fizzbuzz.py::test_entrada_20_deve_retornar_Buzz PASSED [100%]

===== 20 passed in 0.05s =====
Leo@leo:~/Desktop/unidade01/04 - TDD$

```

Fonte: do autor, 2022.

Verifique que todos os casos de testes passaram, e além disso, todas as condições impostas pelo problema FizzBuzz foram tratadas pela função

fizzbuzz. É possível realizar mais testes, porém é desnecessário testar para valores maiores que 20. Outros testes podem incluir tratamento de exceções, valores inválidos e tipos inválidos. Mas isso fica como exercício para completar este exemplo.

Considerações adicionais

A prática do TDD é uma sugestão de boas práticas! Obviamente, que o programador não é obrigado a programar desta maneira. É necessário que o programador acostume-se com esta nova abordagem de programação.

Outro ponto importante a se destacar é que o TDD é uma técnica de programação, e não uma técnica de testes de software. Ela ajuda no desenvolvimento de uma nova funcionalidade por construir testes que guiam o desenvolvimento do código.

Vamos praticar?

1) Suponha que você foi contratado para desenvolver um programa que receba como entrada a idade de um nadador e classifica em uma das seguintes categorias.

Categoria	Idade
Infantil A	5 a 7 anos
Infantil B	8 a 10 anos
Juvenil A	11 a 13 anos
Juvenil B	14 a 17 anos
Sênior	<u>maiores de 18 anos.</u>

- a) Escreva uma função chamada **obter_categoria()** que receba como parâmetro a idade do atleta e devolve a categoria.
- b) Construa o planejamento dos casos de testes determinando os valores de entrada, e valores de saída esperados.
- c) Seguindo a técnica de TDD, escreva os testes automatizados para cada caso de teste definido no item b.

Referências

PYTEST. **Documentação versão de python 3.7+**. 2015. Disponível em: <<https://docs.pytest.org/en/7.1.x/index.html>>. Acesso em: 11 jun. 2022.

SALE, D. **Testing python:** applying unit testing, TDD, BDD, and accepting testing. Wiley, 2014.