

**Exercício2.py:** 100%

38

0

1



```
1 import unittest
2 from Exercício1 import LimpezaCPF, ValidadorCPF
3
4 class TestLimpezaCPF(unittest.TestCase):
5     """
6     Testa se o método limpeza de LimpezaCPF remove corretamente os caracteres
7     não numéricos.
8     """
9     def test_limpeza(self):
10         self.assertEqual(LimpezaCPF.limpeza("123.456.789-00"), "12345678900")
11         self.assertEqual(LimpezaCPF.limpeza("000.111.222-33"), "00011122233")
12         self.assertEqual(LimpezaCPF.limpeza("abc123def456"), "123456")
13
14 class TestValidadorCPF(unittest.TestCase):
15     """
16     Testa se o método _calcular_digito calcula corretamente o dígito verificador.
17     """
18     def test_calcular_digito(self):
19         self.assertEqual(ValidadorCPF._calcular_digito("987654321", 10), 0)
20         self.assertEqual(ValidadorCPF._calcular_digito("9876543210", 11), 10)
21         self.assertEqual(ValidadorCPF._calcular_digito("787159503", 10), 2)
22         self.assertEqual(ValidadorCPF._calcular_digito("7871595032", 11), 8)
23
24     """
25     Testa se o método _obter_digito_verificador obtém corretamente o dígito
26     verificador.
27     """
28     def test_obter_digito_verificador(self):
29         self.assertEqual(ValidadorCPF._obter_digito_verificador("987654321", 10), 0)
30         self.assertEqual(ValidadorCPF._obter_digito_verificador("9876543210", 11), 0)
31         self.assertEqual(ValidadorCPF._obter_digito_verificador("787159503", 10), 2)
```

**Exercício2.py: 100%**

```
34     """
35     Testa se o método _validar_digitos valida corretamente os dígitos
36     verificadores.
37     """
38     def test_validar_digitos(self):
39         self.assertTrue(ValidadorCPF._validar_digitos("98765432100"))
40         self.assertTrue(ValidadorCPF._validar_digitos("78715950328"))
41         self.assertFalse(ValidadorCPF._validar_digitos("00000000000"))
42         self.assertFalse(ValidadorCPF._validar_digitos("498703"))
43
44     """
45     Testa se CPFs válidos passam na validação do método validacao_cpf.
46     """
47     def test_validacao_cpf_valido(self):
48         self.assertTrue(ValidadorCPF("123.456.789-09").validacao_cpf())
49         self.assertTrue(ValidadorCPF("98765432100").validacao_cpf())
50         self.assertTrue(ValidadorCPF("235.998.567-10").validacao_cpf())
51
52     """
53     Testa se se CPFs inválidos falham na validação do método validacao_cpf.
54     """
55     def test_validacao_cpf_invalido(self):
56         self.assertFalse(ValidadorCPF("11111111111").validacao_cpf())
57         self.assertFalse(ValidadorCPF("123.456.789-00").validacao_cpf())
58         self.assertFalse(ValidadorCPF("15698192022").validacao_cpf())
59         self.assertFalse(ValidadorCPF("005.998.567-11").validacao_cpf())
60
61     """
62     Testa se CPFs com formatos inválidos falham na validação do método
63     validacao_cpf.
64     """
65     def test_validacao_cpf_formato_invalido(self):
66         self.assertFalse(ValidadorCPF("123.456.789").validacao_cpf())
67         self.assertFalse(ValidadorCPF("abc123def456").validacao_cpf())
```

**Exercicio2.py: 100%**

```
70
71 | if __name__ == '__main__':
72 |     unittest.main() # pragma: no cover
```

« prev ^ index » next coverage.py v7.3.2, created at 2023-11-30 08:45 -0300