1

```python
import unittest

def is_par(n):
  return n % 2 == 0

class TestIsPar(unittest.TestCase):
  def test_numero_par(self):
    self.assertTrue(is_par(4), "4 deveria ser par")

def test_numero_impar(self):
  self.assertFalse(is_par(7), "7 deveria ser ímpar")

def test_zero(self):
  self.assertTrue(is_par(0), "0 é considerado par")

def test_numero_negativo_par(self):
  self.assertTrue(is_par(-2), "-2 deveria ser par")

def test_numero_negativo_impar(self):
  self.assertFalse(is_par(-3), "-3 deveria ser ímpar")

# Para rodar os testes
# if __name__ == '__main__':
# unittest.main(argv=['first-arg-is-ignored'], exit=False)
```

2

```python
import unittest

def fatorial(n):
    if n < 0:
        raise ValueError("Fatorial não é definido para números
negativos")
    if n == 0:
        return 1
    resultado = 1
    for i in range(1, n + 1):
        resultado *= i
    return resultado

class TestFatorial(unittest.TestCase):
    def test_fatorial_zero(self):
        self.assertEqual(fatorial(0), 1)

    def test_fatorial_positivo(self):
        self.assertEqual(fatorial(5), 120)

    def test_fatorial_um(self):
        self.assertEqual(fatorial(1), 1)

    def test_fatorial_negativo(self):
        with self.assertRaises(ValueError):
            fatorial(-1)

if __name__ == "__main__":
    unittest.main(argv=['first-arg-is-ignored'], exit=False)
```

```
.............
----------------------------------------------------------------------
Ran 13 tests in 0.014s

OK
```

3

```
import unittest

class InsufficientFunds(Exception):
    pass

class Conta:
    def __init__(self, saldo=0):
        self.saldo = saldo

    def depositar(self, amount):
        if amount <= 0:
            raise ValueError("O valor do depósito deve ser positivo")
        self.saldo += amount

    def sacar(self, amount):
        if amount <= 0:
            raise ValueError("O valor do saque deve ser positivo")
        if amount > self.saldo:
            raise InsufficientFunds("Saldo insuficiente")
        self.saldo -= amount

class TestConta(unittest.TestCase):
    def setUp(self):
        self.conta = Conta(100)

    def test_deposito_sucesso(self):
        self.conta.depositar(50)
        self.assertEqual(self.conta.saldo, 150)

    def test_saque_sucesso(self):
        self.conta.sacar(30)
        self.assertEqual(self.conta.saldo, 70)

    def test_saque_insuficiente(self):
        with self.assertRaises(InsufficientFunds):
            self.conta.sacar(200)

    def test_deposito_valor_invalido(self):
        with self.assertRaises(ValueError):
            self.conta.depositar(-10)

    def test_saque_valor_invalido(self):
```

```
        with self.assertRaises(ValueError):
            self.conta.sacar(0)

if __name__ == "__main__":
    unittest.main(argv=['first-arg-is-ignored'], exit=False)
```

```
.........
----------------------------------------------------------------------
Ran 10 tests in 0.012s

OK
```

4

```
import unittest
import requests
from unittest.mock import patch, Mock

class APIError(Exception):
    pass

def buscar_clima(cidade):
    try:
        response =
requests.get(f'https://api.exemplo.com/clima?cidade={cidade}')
        response.raise_for_status()
        dados = response.json()
        if 'temperatura' not in dados:
            raise APIError("Resposta da API não contém a chave
'temperatura'")
        return dados['temperatura']
    except requests.exceptions.RequestException as e:
        raise APIError(f"Erro na requisição: {e}")
    except (ValueError, KeyError) as e:
        raise APIError(f"Erro ao processar JSON: {e}")

class TestBuscarClima(unittest.TestCase):

    @patch('requests.get')
    def test_clima_sucesso(self, mock_get):

        mock_response = Mock()
        mock_response.json.return_value = {'cidade': 'São Paulo',
'temperatura': 25}
        mock_response.raise_for_status.return_value = None
```

```
        mock_get.return_value = mock_response

        temperatura = buscar_clima('São Paulo')
        self.assertEqual(temperatura, 25)

    @patch('requests.get')
    def test_clima_sem_temperatura(self, mock_get):
        mock_response = Mock()
        mock_response.json.return_value = {'cidade': 'São Paulo'}
        mock_response.raise_for_status.return_value = None
        mock_get.return_value = mock_response

        with self.assertRaisesRegex(APIError, "Resposta da API não
contém a chave 'temperatura'"):
            buscar_clima('São Paulo')

    @patch('requests.get')
    def test_api_erro_requerimento(self, mock_get):

        mock_get.side_effect =
requests.exceptions.RequestException('Erro de conexão')

        with self.assertRaisesRegex(APIError, "Erro na requisição"):
            buscar_clima('Qualquer Cidade')



if __name__ == '__main__':
    unittest.main(argv=['first-arg-is-ignored'], exit=False)
```

```
.............
----------------------------------------------------------------------
Ran 13 tests in 0.013s

OK
```