Desenvolvimento de Sistemas II

Prof. Me. Cleber Schroeder Fonseca

Curso Técnico em Informática para Internet
IFRS Rio Grande

2024

Testes

Até aqui usamos o comando assert para fazer uma verificação entre o resultado obtido e o valor esperado.

Mas como conversamos na aula passada o Python possui algumas ferramentas para facilitar a utilização de testes.

Uma das mais usadas (por já vir inclída no código fonte do Python) é a unittest, um framework de teste completo que permite que façamos diversos teste e usemos as mais diversas anotações como veremos a seguir.

UnitTest

O unittest é um módulo de teste integrado ao Python que permite a criação e execução de testes unitários.

Ele fornece uma estrutura para escrever, organizar e executar testes de forma automatizada.

Unittest

Aqui está um exemplo simples de como um teste unitário pode ser escrito em Python, para tanto vamos criar um documento '**test_soma.py**' com o seguinte conteúdo:

```python
import unittest

def soma(a, b):
    return a + b

class TestSoma(unittest.TestCase):
    def test_soma_positiva(self):
        resultado = soma(2, 3)
        self.assertEqual(resultado, 5)  # Verifica se a soma é igual a 5

    def test_soma_negativa(self):
        resultado = soma(-2, -3)
        self.assertEqual(resultado, -5)  # Verifica se a soma é igual a -5

    def test_soma_zero(self):
        resultado = soma(0, 0)
        self.assertEqual(resultado, 0)  # Verifica se a soma é igual a 0

if __name__ == "__main__":
    unittest.main()
```

Unittest

Neste exemplo, temos a função '**soma**' que realiza a adição de dois números. Em seguida, temos uma classe de teste '**TestSoma**' que herda da classe '**unittest.TestCase**', fornecida pelo módulo '**unittest**'.

Dentro dessa classe, temos três métodos de teste, cada um dos quais testa um cenário específico.

Unittest

Os métodos de teste começam com o prefixo '**test_**' para que o framework de teste os reconheça como testes a serem executados.

Dentro de cada método de teste, utilizamos asserções, como '**self.assertEqual**', para verificar se a saída da função '**soma**' corresponde ao resultado esperado.

Unittest

Finalmente, o bloco '**if __name__ == "__main__":**' garante que os testes sejam executados apenas quando o módulo for executado diretamente, não quando importado em outro lugar.

Para executar o teste basta que pela linha de comando:

```
python test_soma.py
```

Tipos de testes unitários

Existem vários tipos de testes unitários em Python que podem ser utilizados para testar o funcionamento correto de componentes individuais do código. Alguns dos tipos mais comuns de testes unitários em Python incluem:

- Testes de asserção (assertion tests): Esses testes verificam se uma determinada condição é verdadeira utilizando a declaração '**assert**' e lançam uma exceção caso a condição seja falsa.
- Testes de casos de borda (boundary tests): Esses testes verificam o comportamento do código nos limites dos valores de entrada ou em situações extremas. Eles ajudam a garantir que o código esteja tratando corretamente os valores extremos e não cause erros ou comportamentos inesperados.
- Testes de exceção (exception tests): Esses testes verificam se uma exceção específica é lançada quando uma determinada entrada ou condição é fornecida ao código. Eles ajudam a garantir que o código esteja gerenciando exceções de maneira adequada.

Tipos de testes unitários

- Testes de casos de uso (use case tests): Esses testes são baseados em cenários específicos de uso do código. Eles verificam se o código está produzindo os resultados esperados em situações reais de uso.
- Testes de mocks: Esses testes utilizam objetos simulados (mocks) para isolar o componente sendo testado de suas dependências externas. Isso permite testar o componente de forma independente e controlada.
- Testes de cobertura de código (code coverage tests): Esses testes são usados para verificar a cobertura do código pelos testes, ou seja, para determinar quais partes do código estão sendo exercitadas pelos testes e quais partes não estão sendo testadas. Isso ajuda a identificar lacunas nos testes e melhorar a qualidade do código.

Asserts

assertEqual

O método '**assertEqual**' é uma função de assertiva fornecida pelo módulo '**unittest**' em Python. Ele verifica se dois valores são iguais e, caso contrário, lança uma exceção indicando a falha do teste.

A sintaxe básica do assertEqual é a seguinte:

```
assertEqual(valor_esperado, valor_obtido, msg=None)
```

Exemplo:

```python
import unittest

def soma(a, b):
    return a + b

class MyTestCase(unittest.TestCase):
    def test_soma(self):
        resultado = soma(2, 2)
        self.assertEqual(resultado, 4)

if __name__ == '__main__':
    unittest.main()
```

Assert

assertNotEqual

O método '**assertNotEqual**' é uma função de assertiva fornecida pelo módulo '**unittest**' em Python. Ele verifica se dois valores são diferentes e, caso contrário, lança uma exceção indicando a falha do teste.

A sintaxe básica do assertNotEqual é a seguinte:

```
assertNotEqual(valor_esperado, valor_obtido, msg=None)
```

Assert

assertNotEqual

Exemplo:

```
1  import unittest
2
3  def subtracao(a, b):
4      return a - b
5
6  class MyTestCase(unittest.TestCase):
7      def test_subtracao(self):
8          resultado = subtracao(5, 3)
9          self.assertNotEqual(resultado, 0)
10
11 if __name__ == '__main__':
12     unittest.main()
```

Assert

assertTrue

O método '**assertTrue**' é uma função de assertiva fornecida pelo módulo '**unittest**' em Python. Ele verifica se uma condição é verdadeira e, caso contrário, lança uma exceção indicando a falha do teste.

A sintaxe básica do assertTrue é a seguinte:

```
assertTrue(condicao, msg=None)
```

Assert

assertTrue

Exemplo:

```
1 import unittest
2
3 class MyTestCase(unittest.TestCase):
4     def test_numero_positivo(self):
5         numero = 5
6         self.assertTrue(numero > 0)
7
8 if __name__ == '__main__':
9     unittest.main()
```

Assert

assertNotTrue

O método '**assertNotTrue**' é uma função de assertiva fornecida pelo módulo '**unittest**' em Python. Ele verifica se uma condição é falsa e, caso contrário, lança uma exceção indicando a falha do teste.

A sintaxe básica do assertNotTrue é a seguinte:

```
assertNotTrue(condicao, msg=None)
```

Assert

assertNotTrue

Exemplo:

```
1 import unittest
2
3 class MyTestCase(unittest.TestCase):
4     def test_numero_negativo(self):
5         numero = -5
6         self.assertNotTrue(numero > 0)
7
8 if __name__ == '__main__':
9     unittest.main()
```

Assert

assertIs

O método '**assertIs**' é uma função de assertiva fornecida pelo módulo '**unittest**' em Python. Ele verifica se dois objetos são idênticos (ou seja, referem-se ao mesmo objeto na memória) e, caso contrário, lança uma exceção indicando a falha do teste.

A sintaxe básica do assertIs é a seguinte:

```
assertIs(objeto_esperado, objeto_obtido, msg=None)
```

Assert

assertIs

Exemplo:

```
1  import unittest
2
3  class MyTestCase(unittest.TestCase):
4      def test_objetos_iguais(self):
5          lista1 = [1, 2, 3]
6          lista2 = lista1
7          self.assertIs(lista1, lista2)
8
9  if __name__ == '__main__':
10     unittest.main()
```

Assert

assertIsNot

O método '**assertIsNot**' é uma função de assertiva fornecida pelo módulo '**unittest**' em Python. Ele verifica se dois objetos não são idênticos (ou seja, não referem-se ao mesmo objeto na memória) e, caso contrário, lança uma exceção indicando a falha do teste.

A sintaxe básica do assertIsNot é a seguinte:

```
assertIsNot(objeto_esperado, objeto_obtido, msg=None)
```

Assert

assertIsNot

Exemplo:

```
1  import unittest
2
3  class MyTestCase(unittest.TestCase):
4      def test_objetos_diferentes(self):
5          lista1 = [1, 2, 3]
6          lista2 = [1, 2, 3]
7          self.assertIsNot(lista1, lista2)
8
9  if __name__ == '__main__':
10     unittest.main()
```

Assert

assertIsNone

O método '**assertIsNone**' é uma função de assertiva fornecida pelo módulo '**unittest**' em Python. Ele verifica se um objeto é '**None**', ou seja, se não possui nenhum valor atribuído, e, caso contrário, lança uma exceção indicando a falha do teste.

A sintaxe básica do assertIsNone é a seguinte:

```
assertIsNone(objeto, msg=None)
```

Assert

assertIsNone

Exemplo:

```
1 import unittest
2
3 class MyTestCase(unittest.TestCase):
4     def test_valor_nulo(self):
5         valor = None
6         self.assertIsNone(valor)
7
8 if __name__ == '__main__':
9     unittest.main()
```

Assert

assertIsNotNone

O método '**assertIsNotNone**' é uma função de assertiva fornecida pelo módulo '**unittest**' em Python. Ele verifica se um objeto não é '**None**', ou seja, se possui algum valor atribuído, e, caso contrário, lança uma exceção indicando a falha do teste.

A sintaxe básica do assertIsNotNone é a seguinte:

```
assertIsNotNone(objeto, msg=None)
```

Assert

assertIsNotNone

Exemplo:

```
1 import unittest
2
3 class MyTestCase(unittest.TestCase):
4     def test_valor_nao_nulo(self):
5         valor = 10
6         self.assertIsNotNone(valor)
7
8 if __name__ == '__main__':
9     unittest.main()
```

Assert

assertIn

O método '**assertIn**' é uma função de assertiva fornecida pelo módulo '**unittest**' em Python. Ele verifica se um determinado valor está presente em uma sequência (como uma lista, tupla, dicionário, etc.) e, caso contrário, lança uma exceção indicando a falha do teste.

A sintaxe básica do assertIn é a seguinte:

```
assertIn(valor, sequencia, msg=None)
```

Assert

assertIn

Exemplo:

```
1 import unittest
2
3 class MyTestCase(unittest.TestCase):
4     def test_valor_presente(self):
5         lista = [1, 2, 3, 4, 5]
6         self.assertIn(3, lista)
7
8 if __name__ == '__main__':
9     unittest.main()
```

Assert

assertNotIn

O método '**assertNotIn**' é uma função de assertiva fornecida pelo módulo '**unittest**' em Python. Ele verifica se um determinado valor não está presente em uma sequência (como uma lista, tupla, conjunto, etc.) e, caso contrário, lança uma exceção indicando a falha do teste.

A sintaxe básica do assertNotIn é a seguinte:

```
assertNotIn(valor, sequencia, msg=None)
```

Assert

assertNotIn

Exemplo:

```
1 import unittest
2
3 class MyTestCase(unittest.TestCase):
4     def test_valor_nao_presente(self):
5         lista = [1, 2, 3, 4, 5]
6         self.assertNotIn(6, lista)
7
8 if __name__ == '__main__':
9     unittest.main()
```

Assert

assertIsInstance

O método '**assertIsInstance**' é uma função de assertiva fornecida pelo módulo '**unittest**' em Python. Ele verifica se um objeto é uma instância de uma determinada classe ou tipo, e, caso contrário, lança uma exceção indicando a falha do teste.

A sintaxe básica do assertIsInstance é a seguinte:

```
assertIsInstance(objeto, tipo, msg=None)
```

Assert

assertIsInstance

Exemplo:

```
1 import unittest
2
3 class MyTestCase(unittest.TestCase):
4     def test_instancia_str(self):
5         valor = "Hello, World!"
6         self.assertIsInstance(valor, str)
7
8 if __name__ == '__main__':
9     unittest.main()
```

Assert

assertNotIsInstance

O método '**assertNotIsInstance**' é uma função de assertiva fornecida pelo módulo '**unittest**' em Python. Ele verifica se um objeto não é uma instância de uma determinada classe ou tipo, e, caso contrário, lança uma exceção indicando a falha do teste.

A sintaxe básica do assertNotIsInstance é a seguinte:

```
assertNotIsInstance(objeto, tipo, msg=None)
```

Assert

assertIsInstance

Exemplo:

```
1 import unittest
2
3 class MyTestCase(unittest.TestCase):
4     def test_nao_instancia_int(self):
5         valor = "Hello, World!"
6         self.assertNotIsInstance(valor, int)
7
8 if __name__ == '__main__':
9     unittest.main()
```

Assert

assertRaises

O método '**assertRaises**' é uma função de assertiva fornecida pelo módulo '**unittest**' em Python. Ele verifica se uma exceção específica é lançada durante a execução de um determinado bloco de código, e, caso contrário, lança uma exceção indicando a falha do teste.

A sintaxe básica do assertRaises é a seguinte:

```
assertRaises(tipoExcecao, funcao, argumentos)
```

Exemplo:

```python
import unittest

def dividir(a, b):
    if b == 0:
        raise ZeroDivisionError("Divisão por zero não é permitida.")
    return a / b

class MyTestCase(unittest.TestCase):
    def test_divisao_por_zero(self):
        with self.assertRaises(ZeroDivisionError):
            dividir(10, 0)

if __name__ == '__main__':
    unittest.main()
```

O método '**assertRaisesRegex**' é uma função de assertiva fornecida pelo módulo '**unittest**' em Python. Ele verifica se uma exceção específica é lançada durante a execução de um determinado bloco de código e se a mensagem de erro da exceção corresponde a um padrão de expressão regular, e, caso contrário, lança uma exceção indicando a falha do teste.

A sintaxe básica do assertRaisesRegex é a seguinte:

```
assertRaisesRegex(tipoExcecao, padrao_regex, funcao, argumentos)
```

Exemplo:

```python
import unittest

def dividir(a, b):
    if b == 0:
        raise ZeroDivisionError("Divisão por zero não é permitida.")
    return a / b

class MyTestCase(unittest.TestCase):
    def test_divisao_por_zero(self):
        with self.assertRaisesRegex(ZeroDivisionError, r"Divisão por zero"):
            dividir(10, 0)

if __name__ == '__main__':
    unittest.main()
```

Assert

assertWarns

O método '**assertWarns**' é uma função de assertiva fornecida pelo módulo '**unittest**' em Python. Ele verifica se um aviso específico é emitido durante a execução de um determinado bloco de código, e, caso contrário, lança uma exceção indicando a falha do teste.

A sintaxe básica do assertWarns é a seguinte:

```
assertWarns(tipoAviso, funcao, argumentos)
```

Exemplo:

```
1  import unittest
2
3  def emitir_aviso():
4      import warnings
5      warnings.warn("Este é um aviso!")
6
7  class MyTestCase(unittest.TestCase):
8      def test_emitir_aviso(self):
9          with self.assertWarns(UserWarning):
10             emitir_aviso()
11
12 if __name__ == '__main__':
13     unittest.main()
```

O método '**assertWarnsRegex**' é uma função de assertiva fornecida pelo módulo '**unittest**' em Python. Ele verifica se um aviso específico é emitido durante a execução de um determinado bloco de código e se a mensagem de aviso corresponde a um padrão de expressão regular, e, caso contrário, lança uma exceção indicando a falha do teste.

A sintaxe básica do assertWarnsRegex é a seguinte:

```
assertWarnsRegex(tipoAviso, padrao_regex, funcao, argumentos)
```

Exemplo:

```python
import unittest

def emitir_aviso():
    import warnings
    warnings.warn("Aviso: Isso é um exemplo!")

class MyTestCase(unittest.TestCase):
    def test_emitir_aviso(self):
        with self.assertWarnsRegex(UserWarning, r'Aviso: .*'):
            emitir_aviso()

if __name__ == '__main__':
    unittest.main()
```

Assert

assertLogs

O método '**assertLogs**' é uma função de assertiva fornecida pelo módulo '**unittest**' em Python. Ele verifica se determinadas mensagens de log são registradas durante a execução de um determinado bloco de código, e, caso contrário, lança uma exceção indicando a falha do teste.

A sintaxe básica do assertLogs é a seguinte:

```
assertLogs(logger, level)
```

Exemplo:

```python
import logging
import unittest

def fazer_algo():
    logging.debug("Isso é uma mensagem de debug")
    logging.info("Isso é uma mensagem de informação")
    logging.warning("Isso é uma mensagem de aviso")

class MyTestCase(unittest.TestCase):
    def test_fazer_algo(self):
        with self.assertLogs(level=logging.WARNING) as logs:
            fazer_algo()
        self.assertIn("Isso é uma mensagem de aviso", logs.output)
        self.assertNotIn("Isso é uma mensagem de debug", logs.output)
        self.assertNotIn("Isso é uma mensagem de informação", logs.output)

if __name__ == '__main__':
    unittest.main()
```

1. Testes
2. Unittest
3. Tipos de testes unitários
4. Asserts
   - Exceções, avisos e logs
   - Verificações específicas
   - verificações por tipo

Assert

assertAlmostEqual

O método '**assertAlmostEqual**' é uma função de assertiva fornecida pelo módulo '**unittest**' em Python. Ele é comumente usado em testes unitários para lidar com cálculos que envolvem números de ponto flutuante, onde pequenas imprecisões podem ocorrer.

A sintaxe básica do assertAlmostEqual é a seguinte:

```
assertAlmostEqual(valor_esperado, valor_obtido, places, msg=None)
```

Assert

assertAlmostEqual

Exemplo:

```
1 import unittest
2
3 class MyTestCase(unittest.TestCase):
4     def test_calculo_preciso(self):
5         resultado = 0.1 + 0.2
6         self.assertAlmostEqual(resultado, 0.3, places=7)
7
8 if __name__ == '__main__':
9     unittest.main()
```

Assert

assertNotAlmostEqual

O método '**assertNotAlmostEqual**' é semelhante ao '**assertAlmostEqual**', mas é usado para verificar se dois valores numéricos não são aproximadamente iguais dentro de uma determinada precisão. Ele também é comumente usado em testes unitários para lidar com cálculos envolvendo números de ponto flutuante.

A sintaxe básica do assertNotAlmostEqual é a seguinte:

```
assertNotAlmostEqual(valor_esperado, valor_obtido, places, msg=None)
```

Assert

assertNotAlmostEqual

Exemplo:

```
1 import unittest
2
3 class MyTestCase(unittest.TestCase):
4     def test_calculo_impreciso(self):
5         resultado = 0.1 + 0.2
6         self.assertNotAlmostEqual(resultado, 0.4, places=7)
7
8 if __name__ == '__main__':
9     unittest.main()
```

Assert

assertGreater

O método '**assertGreater**' é usado para verificar se um valor é maior do que outro. Ele é uma função disponível no módulo '**unittest**' em Python e é comumente usado em testes unitários para comparar valores.

A sintaxe básica do assertGreater é a seguinte:

```
assertGreater(valor1, valor2, msg=None)
```

Assert

assertGreater

Exemplo:

```
1  import unittest
2
3  class MyTestCase(unittest.TestCase):
4      def test_comparacao_numerica(self):
5          numero1 = 5
6          numero2 = 3
7          self.assertGreater(numero1, numero2)
8
9  if __name__ == '__main__':
10     unittest.main()
```

Assert

assertGreaterEqual

O método '**assertGreaterEqual**' é usado para verificar se um valor é maior ou igual a outro. Ele é uma função disponível no módulo '**unittest**' em Python e é comumente usado em testes unitários para comparar valores.

A sintaxe básica do assertGreaterEqual é a seguinte:

```
assertGreaterEqual(valor1, valor2, msg=None)
```

Assert

assertGreaterEqual

Exemplo:

```
1  import unittest
2
3  class MyTestCase(unittest.TestCase):
4      def test_comparacao_numerica(self):
5          numero1 = 5
6          numero2 = 3
7          self.assertGreaterEqual(numero1, numero2)
8          self.assertGreaterEqual(numero1, 5)
9
10 if __name__ == '__main__':
11     unittest.main()
```

Assert

assertLess

O método '**assertLess**' é usado para verificar se um valor é menor do que outro. Ele é uma função disponível no módulo '**unittest**' em Python e é comumente usado em testes unitários para comparar valores.

A sintaxe básica do assertLess é a seguinte:

```
assertLess(valor1, valor2, msg=None)
```

Exemplo:

```
1  import unittest
2
3  class MyTestCase(unittest.TestCase):
4      def test_comparacao_numerica(self):
5          numero1 = 3
6          numero2 = 5
7          self.assertLess(numero1, numero2)
8
9  if __name__ == '__main__':
10     unittest.main()
```

Assert

assertLessEqual

O método '**assertLessEqual**' é usado para verificar se um valor é menor ou igual a outro. Ele é uma função disponível no módulo '**unittest**' em Python e é comumente usado em testes unitários para comparar valores.

A sintaxe básica do assertLessEqual é a seguinte:

```
assertLessEqual(valor1, valor2, msg=None)
```

Exemplo:

```
1  import unittest
2
3  class MyTestCase(unittest.TestCase):
4      def test_comparacao_numerica(self):
5          numero1 = 3
6          numero2 = 5
7          self.assertLessEqual(numero1, numero2)
8          self.assertLessEqual(numero1, 3)
9
10 if __name__ == '__main__':
11     unittest.main()
```

Assert

assertRegex

O método '**assertRegex**' é usado para verificar se uma string corresponde a um padrão de expressão regular. Ele é uma função disponível no módulo '**unittest**' em Python e é comumente usado em testes unitários para verificar se um valor possui um formato esperado.

A sintaxe básica do assertRegex é a seguinte:

```
assertRegex(string, padrao_regex, msg=None)
```

Exemplo:

```python
import unittest

class MyTestCase(unittest.TestCase):
    def test_validacao_string(self):
        email = 'usuario@example.com'
        self.assertRegex(email, r'^\w+@\w+\.\w+$')

        telefone = '1234567890'
        self.assertRegex(telefone, r'^\d{10}$')

if __name__ == '__main__':
    unittest.main()
```

Assert

assertNotRegex

O método '**assertNotRegex**' é usado para verificar se uma string não corresponde a um padrão de expressão regular. Ele é uma função disponível no módulo '**unittest**' em Python e é comumente usado em testes unitários para verificar se um valor não possui um formato específico.

A sintaxe básica do assertNotRegex é a seguinte:

```
assertNotRegex(string, padrao_regex, msg=None)
```

Assert

assertNotRegex

Exemplo:

```python
import unittest

class MyTestCase(unittest.TestCase):
    def test_validacao_string(self):
        email = 'usuario@example.com'
        self.assertNotRegex(email, r'^\d+$')

        telefone = '1234567890'
        self.assertNotRegex(telefone, r'^\D+$')

if __name__ == '__main__':
    unittest.main()
```

Assert

assertCountEqual

O método '**assertCountEqual**' é usado para verificar se duas sequências contêm os mesmos elementos, independentemente da ordem. Ele é uma função disponível no módulo '**unittest**' em Python e é comumente usado em testes unitários para comparar a igualdade de duas listas, tuplas ou outros tipos de sequências.

A sintaxe básica do assertCountEqual é a seguinte:

```
assertCountEqual(seq1, seq2, msg=None)
```

Assert

assertCountEqual

Exemplo:

```
1  import unittest
2
3  class MyTestCase(unittest.TestCase):
4      def test_comparacao_sequencias(self):
5          lista1 = [1, 2, 3, 4, 5]
6          lista2 = [5, 4, 3, 2, 1]
7          self.assertCountEqual(lista1, lista2)
8
9          tupla1 = (1, 2, 3)
10         tupla2 = (3, 1, 2)
11         self.assertCountEqual(tupla1, tupla2)
12
13 if __name__ == '__main__':
14     unittest.main()
```

1. Testes
2. Unittest
3. Tipos de testes unitários
4. Asserts
   - Exceções, avisos e logs
   - Verificações específicas
   - verificações por tipo

Assert

assertMultiLineEqual

O método '**assertMultiLineEqual**' é usado para comparar duas strings em várias linhas. Ele é uma função disponível no módulo '**unittest**' em Python e é comumente usado em testes unitários para verificar a igualdade de strings que contêm várias linhas de texto.

A sintaxe básica do assertMultiLineEqual é a seguinte:

```
assertMultiLineEqual(str1, str2, msg=None)
```

Exemplo:

```
1  import unittest
2
3  class MyTestCase(unittest.TestCase):
4      def test_comparacao_strings_multilinhas(self):
5          texto1 = "Olá!\nBem-vindo ao Python.\nTenha um bom dia!"
6          texto2 = "Olá!\nBem-vindo ao Python.\nTenha um bom dia!"
7          self.assertMultiLineEqual(texto1, texto2)
8
9          texto3 = "Lorem ipsum dolor sit amet,\nNullam at rutrum magna."
10         texto4 = "Lorem ipsum dolor sit amet,\nNullam at rutrum magna!"
11         self.assertMultiLineEqual(texto3, texto4)
12
13 if __name__ == '__main__':
14     unittest.main()
```

Assert

assertSequenceEqual

O método '**assertSequenceEqual**' é usado para comparar duas sequências, como listas, tuplas ou outros tipos de sequências, em Python. Ele é uma função disponível no módulo '**unittest**' e é comumente usado em testes unitários para verificar se duas sequências são iguais.

A sintaxe básica do assertSequenceEqual é a seguinte:

```
assertSequenceEqual(seq1, seq2, msg=None)
```

Assert

assertSequenceEqual

Exemplo:

```
1  import unittest
2
3  class MyTestCase(unittest.TestCase):
4      def test_comparacao_sequencias(self):
5          lista1 = [1, 2, 3]
6          lista2 = [1, 2, 3]
7          self.assertSequenceEqual(lista1, lista2)
8
9          tupla1 = (4, 5, 6)
10         tupla2 = (4, 5, 6)
11         self.assertSequenceEqual(tupla1, tupla2)
12
13         conjunto1 = {7, 8, 9}
14         conjunto2 = {9, 7, 8}
15         self.assertSequenceEqual(conjunto1, conjunto2)
16
17 if __name__ == '__main__':
18     unittest.main()
```

Assert

assertListEqual

O método '**assertListEqual**' é usado para comparar duas listas em Python. Ele é uma função disponível no módulo '**unittest**' e é comumente usado em testes unitários para verificar se duas listas são iguais.

A sintaxe básica do assertListEqual é a seguinte:

```
assertListEqual(list1, list2, msg=None)
```

Assert

assertListEqual

Exemplo:

```
1  import unittest
2
3  class MyTestCase(unittest.TestCase):
4      def test_comparacao_listas(self):
5          lista1 = [1, 2, 3]
6          lista2 = [1, 2, 3]
7          self.assertListEqual(lista1, lista2)
8
9          lista3 = ['a', 'b', 'c']
10         lista4 = ['a', 'b', 'c']
11         self.assertListEqual(lista3, lista4)
12
13         lista5 = [True, False, True]
14         lista6 = [True, False, True]
15         self.assertListEqual(lista5, lista6)
16
17 if __name__ == '__main__':
18     unittest.main()
```

Assert

assertTupleEqual

O método '**assertTupleEqual**' é usado para comparar duas tuplas em Python. Ele é uma função disponível no módulo '**unittest**' e é comumente usado em testes unitários para verificar se duas tuplas são iguais.

A sintaxe básica do assertTupleEqual é a seguinte:

```
assertTupleEqual(tuple1, tuple2, msg=None)
```

Assert

assertTupleEqual

Exemplo:

```python
import unittest

class MyTestCase(unittest.TestCase):
    def test_comparacao_tuplas(self):
        tupla1 = (1, 2, 3)
        tupla2 = (1, 2, 3)
        self.assertTupleEqual(tupla1, tupla2)

        tupla3 = ('a', 'b', 'c')
        tupla4 = ('a', 'b', 'c')
        self.assertTupleEqual(tupla3, tupla4)

        tupla5 = (True, False, True)
        tupla6 = (True, False, True)
        self.assertTupleEqual(tupla5, tupla6)

if __name__ == '__main__':
    unittest.main()
```

Assert

assertDictEqual

O método '**assertDictEqual**' é usado para comparar dois dicionários em Python. Ele é uma função disponível no módulo '**unittest**' e é comumente usado em testes unitários para verificar se dois dicionários são iguais.

A sintaxe básica do assertDictEqual é a seguinte:

```
assertDictEqual(dict1, dict2, msg=None)
```

Assert

assertDictEqual

Exemplo:

```python
import unittest

class MyTestCase(unittest.TestCase):
    def test_comparacao_dicionarios(self):
        dicionario1 = {'chave1': 1, 'chave2': 2, 'chave3': 3}
        dicionario2 = {'chave1': 1, 'chave2': 2, 'chave3': 3}
        self.assertDictEqual(dicionario1, dicionario2)

        dicionario3 = {'nome': 'João', 'idade': 25, 'cidade': 'São Paulo'}
        dicionario4 = {'nome': 'João', 'idade': 25, 'cidade': 'São Paulo'}
        self.assertDictEqual(dicionario3, dicionario4)

        dicionario5 = {1: 'a', 2: 'b', 3: 'c'}
        dicionario6 = {1: 'a', 2: 'b', 3: 'c'}
        self.assertDictEqual(dicionario5, dicionario6)

if __name__ == '__main__':
    unittest.main()
```

MUITO OBRIGADO!

Cleber Schroeder Fonseca

http://ifrs.edu.br/riogrande

profcleberfonseca@gmail.com
cleber.fonseca@riogrande.ifrs.edu.br