**Task:**

You are developing a Python utility module that includes a set of mathematical and string-processing functions (e.g., factorial, is\_prime, reverse\_string, normalize\_whitespace). Your goal is to design and implement **comprehensive, maintainable, and readable parameterized test cases** using pytest.

**Ans: utils.py:**

|  |
| --- |
| def factorial(n: int) -> int:  if not isinstance(n, int):  raise TypeError("Input must be an integer")  if n < 0:  raise ValueError("Input must be non-negative")  if n == 0:  return 1  return n \* factorial(n - 1) def is\_prime(n: int) -> bool:  if not isinstance(n, int):  raise TypeError("Input must be an integer")  if n < 1:  raise ValueError("Input must be a positive integer")  if n == 1:  return False  for i in range(2, int(n \*\* 0.5) + 1):  if n % i == 0:  return False  return True def reverse\_string(s: str) -> str:   if not isinstance(s, str):  raise TypeError("Input must be a string")  return s[::-1] def normalize\_whitespace(s: str) -> str:  if not isinstance(s, str):  raise TypeError("Input must be a string")  return ' '.join(s.split()) |

**test\_utils.py:**

|  |
| --- |
| import pytest from utils import factorial, is\_prime, reverse\_string, normalize\_whitespace @pytest.fixture def string\_inputs():  return {  "empty": "",  "single\_char": "a",  "normal": "Hello World",  "with\_spaces": " Multiple Spaces ",  "unicode": "こんにちは" } @pytest.mark.parametrize(  "input\_n, expected\_result, test\_id",  [ (0, 1, "zero"),  (1, 1, "one"),  (5, 120, "positive\_normal"),  (10, 3628800, "large\_positive"),],  ids=lambda x: x[2] ) def test\_factorial\_normal\_cases(input\_n, expected\_result, test\_id):  assert factorial(input\_n) == expected\_result @pytest.mark.parametrize(  "input\_n, expected\_exception, test\_id",  [(-1, ValueError, "negative"),  (3.5, TypeError, "float"),  ("2", TypeError, "string"),],  ids=lambda x: x[2]) def test\_factorial\_error\_cases(input\_n, expected\_exception, test\_id):  with pytest.raises(expected\_exception):  factorial(input\_n) @pytest.mark.parametrize(  "input\_n, expected\_result, test\_id",  [ (2, True, "smallest\_prime"),  (3, True, "odd\_prime"),  (4, False, "small\_non\_prime"),  (17, True, "larger\_prime"),  (100, False, "large\_non\_prime"),  (1, False, "one"),],  ids=lambda x: x[2]) def test\_is\_prime\_normal\_cases(input\_n, expected\_result, test\_id):  assert is\_prime(input\_n) == expected\_result @pytest.mark.parametrize(  "input\_n, expected\_exception, test\_id",  [ (0, ValueError, "zero"),  (-5, ValueError, "negative"),  (2.5, TypeError, "float"),  ("7", TypeError, "string"),],  ids=lambda x: x[2] ) def test\_is\_prime\_error\_cases(input\_n, expected\_exception, test\_id):  with pytest.raises(expected\_exception):  is\_prime(input\_n) @pytest.mark.parametrize(  "input\_key, expected\_output",  [ ("empty", ""),  ("single\_char", "a"),  ("normal", "dlroW olleH"),  ("with\_spaces", " secapS elpitluM "),  ("unicode", "はちにんこ"),],  ids=lambda x: x[0]) def test\_reverse\_string\_normal\_cases(string\_inputs, input\_key, expected\_output):  assert reverse\_string(string\_inputs[input\_key]) == expected\_output @pytest.mark.parametrize(  "input\_val, test\_id",  [  (123, "integer"),  (None, "none"),  ([1, 2, 3], "list"),  ],  ids=lambda x: x[1]) def test\_reverse\_string\_error\_cases(input\_val, test\_id):  with pytest.raises(TypeError):  reverse\_string(input\_val) @pytest.mark.parametrize(  "input\_str, expected\_output",  [ ("", "", "empty"),  ("Hello", "Hello", "single\_word"),  (" Hello World ", "Hello World", "multiple\_spaces"),  ("\t\n Test String \r\n", "Test String", "mixed\_whitespace"),  ("Hello\tWorld", "Hello World", "tabs"),],  ids=lambda x: x[2] if len(x) == 3 else f"case\_{x[0]}" ) def test\_normalize\_whitespace\_normal\_cases(input\_str, expected\_output):  assert normalize\_whitespace(input\_str) == expected\_output @pytest.mark.parametrize(  "input\_val, test\_id",  [ (123, "integer"),  (None, "none"),  ([1, 2, 3], "list"), ],  ids=lambda x: x[1]) def test\_normalize\_whitespace\_error\_cases(input\_val, test\_id):  with pytest.raises(TypeError):  normalize\_whitespace(input\_val) |

**OUTPUT:**

