
1-Pytest - PowerPoint
Tell me what you want to do
Share
File
Home
Insert
Design
Transitions
Animations
Slide Show
Review
View
Help
Paste
Cut
Copy
Format Painter
Clipboard
New Slide
Layout
Reset
Section
Slides
Font
Paragraph
Text Direction
Align Text
Convert to SmartArt
Drawing
Arrange
Quick Styles
Shape Fill
Shape Outline
Shape Effects
Find
Replace
Select
Editing
YASH Technologies
SM
More than what you think.
Python Testing
© 2017 YASH Technologies | www.yash.com | Confidential
Slide 1 of 36
English (India)
Notes
Comments
67%
Search
ENG
IN
14:37
04-11-2025

What is Testing?

Testing is the process of verifying that your code works as expected. It involves writing additional code that checks whether your main code produces the correct results for various inputs and scenarios.

A test **is** a code that checks the validity of the other code

Simple Analogy

Think of testing like taste-testing food while cooking:

- You check if the seasoning is right.
- You verify if it's cooked properly.
- You ensure it meets expectations before serving.

YASH Technologies
More than what you think.

Why Testing is Important

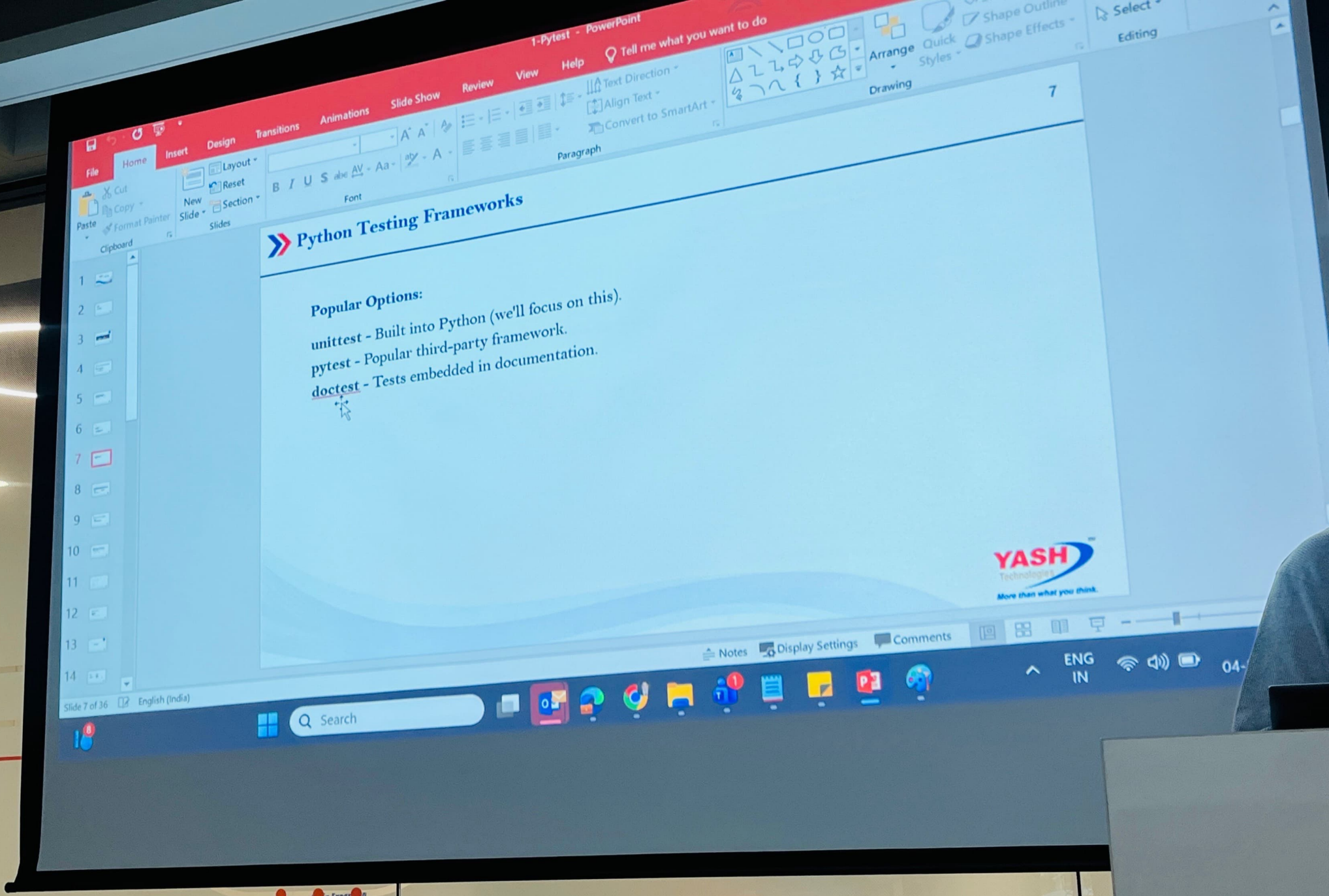
Catch Bugs Early: Find issues before they reach production.
Documentation: Tests serve as living documentation.
Refactoring Safety: Ensure changes don't break existing functionality.
Code Quality: Encourages better, more modular code design.
Confidence: Deploy with confidence knowing your code works.

Types of Testing

1. **Unit Testing**
   - Tests individual units/components in isolation.
   - Fast and focused.
   - Example: Testing a single function.

2. **Integration Testing**
   - Tests how different components work together.
   - Example: Testing database interactions with business logic.

3. **Functional Testing**
   - Tests complete functionality from user perspective.
   - Example: Testing API endpoints.


1-Pytest - PowerPoint
Tell me what you want to do
File
Home
Insert
Design
Transitions
Animations
Slide Show
Review
View
Help
Paste
Cut
Copy
Format Painter
Clipboard
New Slide
Layout
Reset
Section
Slides
Font
Paragraph
Text Direction
Align Text
Convert to SmartArt
Arrange
Quick Styles
Shape Outline
Shape Effects
Drawing
Select
Editing
7
Python Testing Frameworks
Popular Options:
unittest - Built into Python (we'll focus on this).
pytest - Popular third-party framework.
doctest - Tests embedded in documentation.
YASH Technologies
More than what you think.
Notes
Display Settings
Comments
Slide 7 of 36
English (India)
Search
ENG
IN

Difference between python unittest and pytest

Unittest and pytest are both popular testing frameworks in Python, but they differ significantly in their approach and features.

1. Test Structure and Syntax:

unittest: Follows the xUnit style, requiring tests to be written as methods within classes that inherit from unittest.TestCase. Assertions are made using specific methods like self.assertTrue(), self.assertEqual(), etc.

```
import unittest
class TestMath(unittest.TestCase):
  def test_addition(self):
    num = 3
    num2 = 4
    self.assertEqual(num + 1, num2)    # Pass
    self.assertNotEqual(num, num2)     # Pass

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

Run: python -m unittest test_file.py

YASH Technologies
More than what you think.

pytest:

Pytest, Allows tests to be written as plain Python functions, which is generally more concise. Assertions use standard Python assert statements, providing powerful introspection for clearer failure messages.

```
def test_addition():
    num = 3
    num2 = 4
    assert num + 1 == num2, "Addition test passed"
    assert num != num2, "Inequality test passed"
```

Run: pytest -v

YASH Technologies
More than what you think.

2. Test Discovery:

- **unittest:**
  Requires a specific directory structure or manual configuration for test discovery.
- **pytest:**
  Offers automatic test discovery, finding tests in files named test_*.py or *_test.py and functions/methods prefixed with test_.
- **Powerful Test Discovery:** `pytest` has a robust and automatic test discovery mechanism. It can find and run test cases without requiring you to explicitly specify which tests to run, saving you time and effort.

3. Fixtures:

1- unittest:

Manages test setup and teardown using methods like setUp() and tearDown() within TestCase classes.

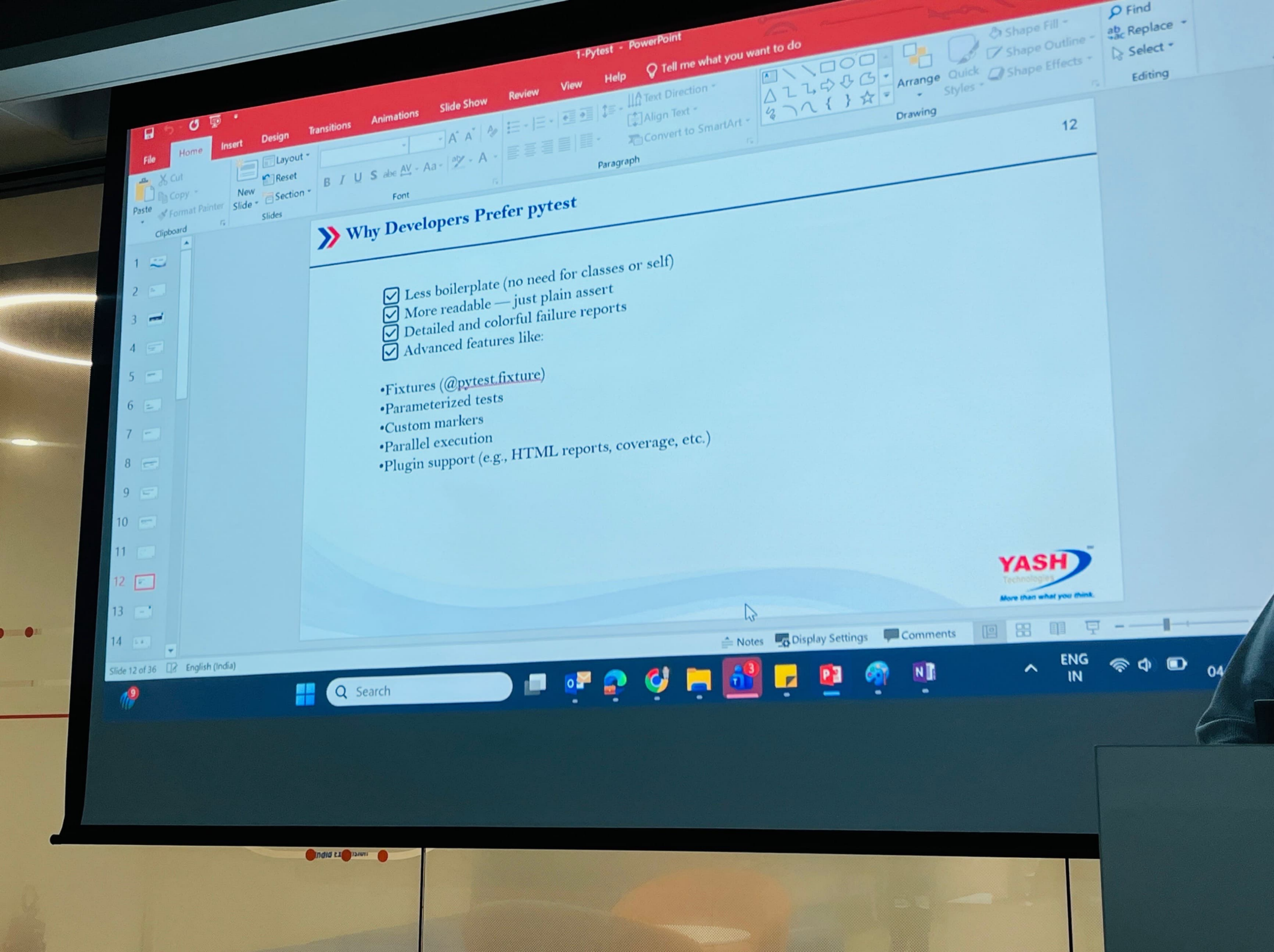
2- pytest:

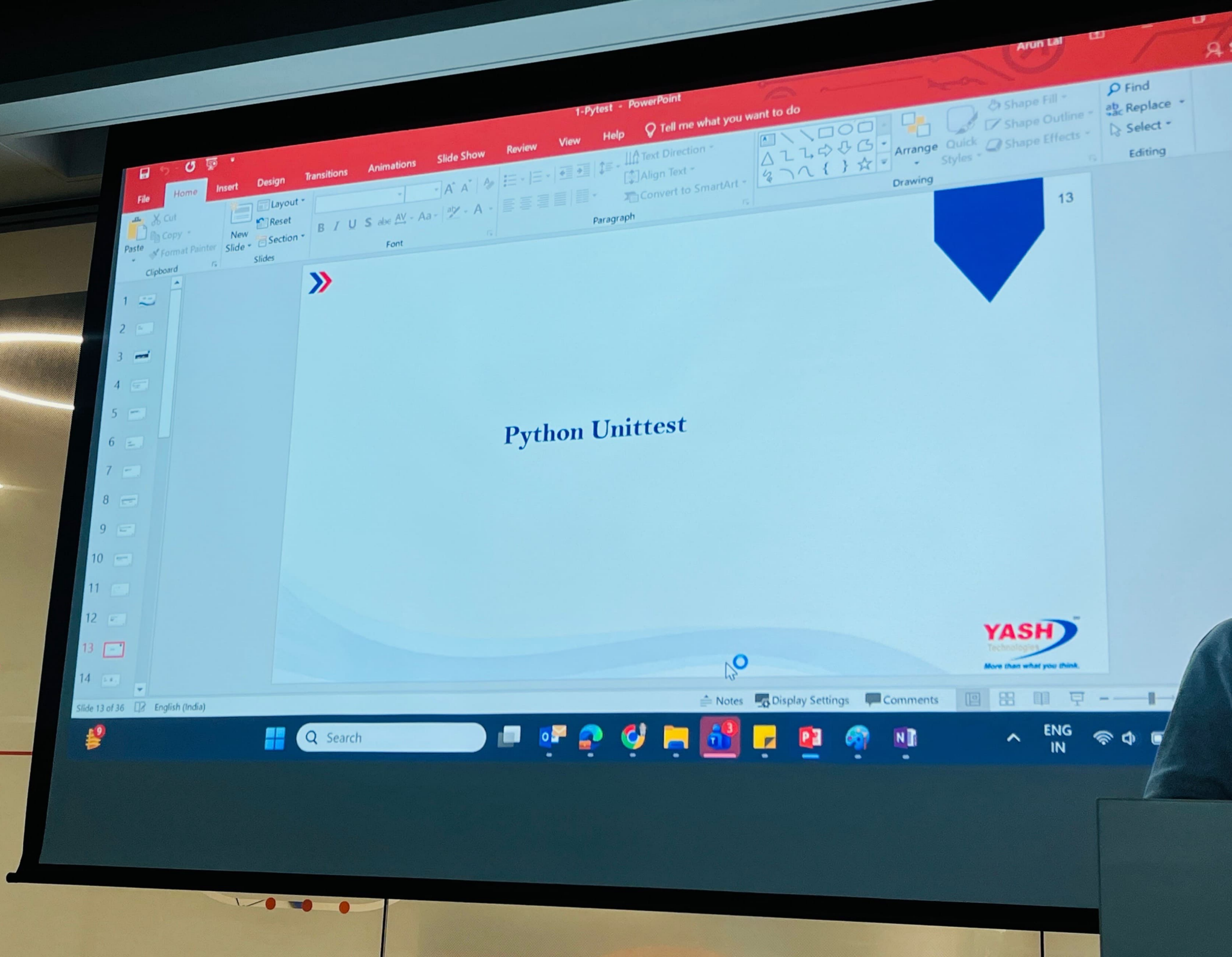
Employs a more powerful and flexible fixture system, where fixtures are functions that can be injected into test functions as arguments, promoting reusability and explicit dependency management.

4. Assertions:

- unittest: Uses specific assertion methods, which can be less readable in some cases.
- pytest: Leverages standard Python assert statements, providing detailed and informative failure messages with introspection, making debugging easier.

YASH Technologies
More than what you think.


1-Pytest - PowerPoint
Tell me what you want to do
Why Developers Prefer pytest
Less boilerplate (no need for classes or self)
More readable — just plain assert
Detailed and colorful failure reports
Advanced features like:
•Fixtures (@pytest.fixture)
•Parameterized tests
•Custom markers
•Parallel execution
•Plugin support (e.g., HTML reports, coverage, etc.)
YASH
Technologies
More than what you think.
Notes
Display Settings
Comments
Slide 12 of 36
English (India)
Search
ENG
IN


1-Pytest - PowerPoint
Tell me what you want to do
File
Home
Insert
Design
Transitions
Animations
Slide Show
Review
View
Help
Clipboard
Slides
Font
Paragraph
Drawing
Editing
13
Python Unittest
YASH
Technologies
Move then what you think.
Slide 13 of 36
English (India)
Notes
Display Settings
Comments
Search
ENG
IN

Using unittest (Built-in Framework)

calculator.py

```
def add(a, b):
    """Add two numbers"""
    return a + b

def subtract(a, b):
    """Subtract b from a"""
    return a - b

def multiply(a, b):
    """Multiply two numbers"""
    return a * b

def divide(a, b):
    """Divide a by b"""
    if b == 0:
        raise ValueError("Cannot divide by zero")
    return a / b
```

Using unittest (Built-in Framework)

```python
# test_calculator_unittest.py
import unittest
from calculator import add, subtract, multiply, divide

class TestCalculator(unittest.TestCase):

    def test_add(self):
        """Test addition function"""
        self.assertEqual(add(2, 3), 5)
        self.assertEqual(add(-1, 1), 0)
        self.assertEqual(add(0, 0), 0)

    def test_subtract(self):
        """Test subtraction function"""
        self.assertEqual(subtract(5, 3), 2)
        self.assertEqual(subtract(3, 5), -2)
        self.assertEqual(subtract(0, 0), 0)
```

YASH Technologies
More than what you think.

Using unittest (Built-in Framework)

```
def test_multiply(self):
    """Test multiplication function"""
    self.assertEqual(multiply(4, 5), 20)
    self.assertEqual(multiply(-2, 3), -6)
    self.assertEqual(multiply(0, 100), 0)

def test_divide(self):
    """Test division function"""
    self.assertEqual(divide(10, 2), 5)
    self.assertEqual(divide(9, 3), 3)
    self.assertAlmostEqual(divide(7, 3), 2.333, places=3)

def test_divide_by_zero(self):
    """Test division by zero raises error"""
    with self.assertRaises(ValueError):
        divide(10, 0)

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

Using unittest (Built-in Framework)

```python
# test_calculator_unittest.py
import unittest
from calculator import add, subtract, multiply, divide

class TestCalculator(unittest.TestCase):

    def test_add(self):
        """Test addition function"""
        self.assertEqual(add(2, 3), 5)
        self.assertEqual(add(-1, 1), 0)
        self.assertEqual(add(0, 0), 0)

    def test_subtract(self):
        """Test subtraction function"""
        self.assertEqual(subtract(5, 3), 2)
        self.assertEqual(subtract(3, 5), -2)
        self.assertEqual(subtract(0, 0), 0)
```

YASH Technologies
More than what you think.