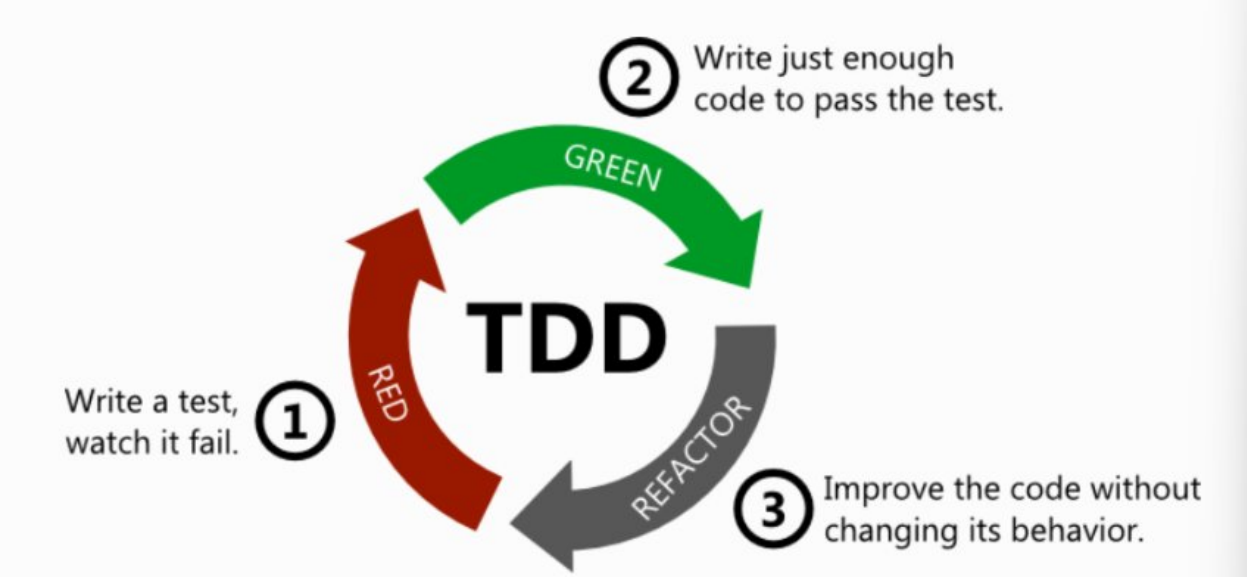
Test-driven development

Test-driven development (TDD) is a software development methodology where the developer writes automated tests before writing code. This approach helps ensure that the code meets the requirements and behaves as expected.

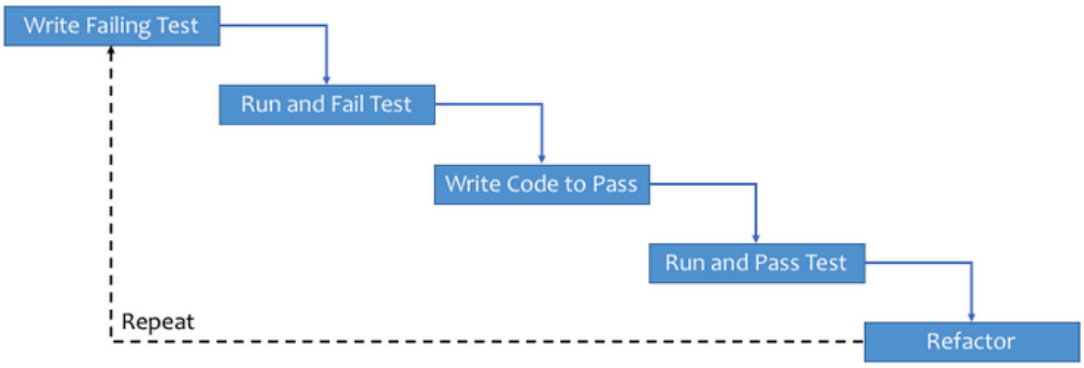
Benefits of Test-Driven Development

1. **Usable Software**: TDD prioritizes usability, leading to software that meets user needs effectively.
2. **Bug Reduction**: Early bug detection results in more solid and error-free software.
3. **Functionality Identification**: TDD quickly identifies functionality issues, enabling rapid resolution.
4. **Code Quality**: Constant refactoring and testing ensure a clean and high-quality codebase.
5. **Code Simplification**: TDD promotes code simplicity, making the codebase streamlined and easier to maintain.
6. **Quality Check**: Tests serve as quality metrics, allowing developers to assess and improve code quality.
7. **Extensible Software**: TDD encourages modular development, resulting in flexible and easily extendable software.
8. **Documentation**: Tests serve as up-to-date documentation, aiding other developers in understanding the codebase.

TDD Cycle



Steps With More detail



Example for Calculator:

1-Writing Tests

[37]:
```
import unittest
```

[38]:
```
import math

def add(a, b):
    pass

def subtract(a, b):
    pass

def multiply(a, b):
    pass

def divide(a, b):
    pass

def power(a, b):
    pass

def square_root(a):
    pass
```

[52]:
```python
class TestCalculator(unittest.TestCase):

    def test_add(self):
        self.assertEqual(add(3, 5), 8)

    def test_subtract(self):
        self.assertEqual(subtract(5, 3), 2)

    def test_multiply(self):
        self.assertEqual(multiply(3, 5), 15)

    def test_divide(self):
        self.assertEqual(divide(3, 1), 3)

    def test_power(self):
        self.assertEqual(power(2, 3), 8)

    def test_square_root(self):
        self.assertEqual(square_root(9), 3)
```

2-Run & Watch It Fail:

[40]:
```python
suite = unittest.TestLoader().loadTestsFromTestCase(TestCalculator)
unittest.TextTestRunner().run(suite)
```

***Note**

`unittest.TestLoader().loadTestsFromTestCase(TestCalculator)` method is responsible for loading all the test methods defined in the `TestCalculator` class and adding them to the test suite (`suite`).

***Note**

`unittest.TextTestRunner()` is used to run tests and display the results in a human-readable format.

```
FFFFFF
======================================================================
FAIL: test_add (__main__.TestCalculator)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\mena\AppData\Local\Temp\ipykernel_14392\3183850598.py", line 4, in test_add
    self.assertEqual(add(3, 5), 8)
AssertionError: None != 8

======================================================================
FAIL: test_divide (__main__.TestCalculator)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\mena\AppData\Local\Temp\ipykernel_14392\3183850598.py", line 13, in test_divide
    self.assertEqual(divide(3, 1), 3)
AssertionError: None != 3

======================================================================
FAIL: test_multiply (__main__.TestCalculator)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\mena\AppData\Local\Temp\ipykernel_14392\3183850598.py", line 10, in test_multiply
    self.assertEqual(multiply(3, 5), 15)
AssertionError: None != 15

======================================================================
FAIL: test_power (__main__.TestCalculator)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\mena\AppData\Local\Temp\ipykernel_14392\3183850598.py", line 16, in test_power
    self.assertEqual(power(2, 3), 8)
AssertionError: None != 8

======================================================================
FAIL: test_square_root (__main__.TestCalculator)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\mena\AppData\Local\Temp\ipykernel_14392\3183850598.py", line 19, in test_square_root
    self.assertEqual(square_root(9), 3)
AssertionError: None != 3

======================================================================
FAIL: test_subtract (__main__.TestCalculator)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\mena\AppData\Local\Temp\ipykernel_14392\3183850598.py", line 7, in test_subtract
    self.assertEqual(subtract(5, 3), 2)
AssertionError: None != 2

----------------------------------------------------------------------
Ran 6 tests in 0.043s

FAILED (failures=6)
```

```
]: <unittest.runner.TextTestResult run=6 errors=0 failures=6>
```

3-Write The Code

[59]:
```python
import math

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

def subtract(a, b):
    return a - b

def multiply(a, b):
    return a * b

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

def power(a, b):
    return a ** b

def square_root(a):
    if a < 0:
        raise ValueError("Cannot calculate square root of a negative number!")
    return math.sqrt(a)
```

[60]:
```python
class TestCalculator(unittest.TestCase):

    def test_add(self):
        self.assertEqual(add(3, 5), 8)
        self.assertNotEqual(add(3, 5), 7)
        self.assertAlmostEqual(add(0.1, 0.2), 0.3, places=5)

    def test_subtract(self):
        self.assertEqual(subtract(5, 3), 2)
        self.assertNotEqual(subtract(5, 3), 3)
        self.assertAlmostEqual(subtract(0.3, 0.1), 0.2, places=5)

    def test_multiply(self):
        self.assertEqual(multiply(3, 5), 15)
        self.assertNotEqual(multiply(3, 5), 12)
        self.assertAlmostEqual(multiply(0.1, 0.2), 0.02, places=5)

    def test_divide(self):
        self.assertEqual(divide(10, 2), 5)
        self.assertNotEqual(divide(10, 2), 4)
        self.assertAlmostEqual(divide(1, 3), 0.33333, places=5)
        with self.assertRaises(ValueError):
            divide(10, 0)

    def test_power(self):
        self.assertEqual(power(2, 3), 8)
        self.assertNotEqual(power(2, 3), 7)
        self.assertAlmostEqual(power(2, 0.5), 1.41421, places=5)

    def test_square_root(self):
        self.assertEqual(square_root(9), 3)
        self.assertNotEqual(square_root(9), 4)
        self.assertAlmostEqual(square_root(2), 1.41421, places=5)
        with self.assertRaises(ValueError):
            square_root(-1)
```

4- Run & Pass The Test

[61]:
```
# Create a test suite
suite = unittest.TestLoader().loadTestsFromTestCase(TestCalculator)

# Run the test suite
unittest.TextTestRunner().run(suite)
```

```
......
----------------------------------------------------------------------
Ran 6 tests in 0.011s

OK
```

[61]:
```
<unittest.runner.TextTestResult run=6 errors=0 failures=0>
```

5- Refactor

```
def add(a, b):
    return a + b
```

=====>

```
def add(*args):
    return sum(args)
```