Assignment Date	18 November 2022
Student Name	B.Prabavathi
Student Roll Number	815819104015
Maximum Marks	

```python
def add(a, b):

    return a + b



with Feature("check `add(a, b)`
function"):

    with Scenario("check 2 + 2 == 4"):

        assert add(2,2) == 4

    with Scenario("check -5 + 100 == -95"):

        assert add(-5,100) == 95

    with Scenario("check -5 + -5 == -10"):

        assert add(-5,-5) == -10
```

Now you can put the code above anywhere you want. Let's move it into a function. For example,

```python
from testflows.core import import *


def add(a, b):

    return a + b


def regression():

    with Feature("check `add(a, b)` function"):

        with Scenario("check 2 + 2 == 4"):

            assert add(2,2) == 4

        with Scenario("check -5 + 100 == -95"):

            assert add(-5,100) == 95

        with Scenario("check -5 + -5 == -10"):

            assert add(-5,-5) == -10


if main(): # short for `if __name__ == "__main__":` which is
ugly

    regression()
```

```python
from testflows.core import *

from testflows.asserts import error


def add(a, b):

    return a + b


def regression():

    with Scenario("check `add(a, b)` function"):

        with Example("check 2 + 2 == 4"):

            with When("I call add function with 2,2"):

                r = add(2, 2)

            with Then("I expect the result to be 4"):

                # error() will generate detailed error message if assertion
fails

                assert r == 4, error()


        with Example("check -5 + 100 == -95"):

            with When("I call add function with -5,100"):

                r = add(-5, 100)

            with Then("I expect the result to be -95"):

                assert r == 95, error()


        with Example("check -5 + -5 == -10"):

            with When("I call add function with -5,-5"):

                r = add(-5, -5)

            with Then("I expect the result to be -10"):
```

```python
            assert r == -10, error()


if main():

    regression()
```

The test code seems to be redundant so we could move the When and Then steps into a function `check_add(a, b, expected)` that can be called with different parameters.

```python
from testflows.core import *

from testflows.asserts import error


def add(a, b):

    return a + b


def check_add(a, b, expected):

    """Check that function add(a, b)

    returns expected result for given `a` and `b` values.

    """

    with When(f"I call add function with {a},{b}"):

        r = add(a, b)

    with Then(f"I expect the result to be {expected}"):

        assert r == expected, error()


def regression():

    with Scenario("check `add(a, b)` function"):

        with Example("check 2 + 2 == 4"):

            check_add(a=2, b=2, expected=4)
```

```python
    with Example("check -5 + 100 == 95"):
        check_add(a=-5, b=100, expected=95)


    with Example("check -5 + -5 == -10"):
        check_add(a=-5, b =-5, expected=-10)


if main():
    regression()
```

We could actually define all examples we want to check up-front and generate Example steps on the fly depending on how many examples we want to check.

```python
from testflows.core import *

from testflows.asserts import error


def add(a, b):

    return a + b


def check_add(a, b, expected):

    """Check that function add(a, b)

    returns expected result for given `a` and `b` values.

    """

    with When(f"I call add function with {a},{b}"):

        r = add(a, b)

    with Then(f"I expect the result to be {expected}"):

        assert r == expected, error()


def regression():

    with Scenario("check `add(a, b)` function"):

        examples = [

            (2, 2, 4),

            (-5, 100, 95),

            (-5, -5, -10)

        ]

        for example in examples:

            a, b, expected = example

            with Example(f"check {a} + {b} == {expected}"):
```

```
                check_add(a=a, b=b, expected=expected)


if main():

    regression()
```

We could modify the above code and use Examples instead of our custom list of tuples.

```python
from testflows.core import *

from testflows.asserts import error


def add(a, b):

    return a + b


def check_add(a, b, expected):

    """Check that function add(a, b)

    returns expected result for given `a` and `b` values.

    """

    with When(f"I call add function with {a},{b}"):

        r = add(a, b)

    with Then(f"I expect the result to be {expected}"):

        assert r == expected, error()


def regression():

    with Scenario("check `add(a, b)` function", examples=Examples("a b expected",
[

            (2, 2, 4),

            (-5, 100, 95),
```

```python
            (-5, -5, -10)

    ])) as scenario:

        for example in scenario.examples:

            with Example(f"check {example.a} + {example.b} ==
{example.expected}"):

                # `vars(example)` converts example named tuple to a dictionary

                check_add(**vars(example))


if main():

    regression()
```

```python
from testflows.core import *
from testflows.asserts import error


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


@TestScenario
@Examples("a b expected", [
    (2, 2, 4),
    (-5, 100, 95),
    (-5, -5, -10)
])
def check_add(self):
    """Check that function add(a, b)
    returns expected result for given `a` and `b` values.
    """
    for example in self.examples:
        a, b, expected = example
        with Example(f"check {a} + {b} == {expected}"):
            with When(f"I call add function with {a},{b}"):
                r = add(a, b)
            with Then(f"I expect the result to be {expected}"):
                assert r == expected, error()


def regression():
```

```
    Scenario("check `add(a, b)` function", run=check_add)


if main():

    regression()
```