writing-test-cases-I.md

Writing Test Cases I

`gumball_machine.py`

```
class GumballMachine:
    def __init__(self):
        self.valid_currency = {"nickel": 5, "dime": 10, "quarter": 25}
        # Represents the total value of valid currency the user inserted
        self.currency_value = 0
        self.invalid_currencies = []

    def insert(self, currency):
        # Checks if the inserted value is a valid currency, then adds it
        if currency in self.valid_currency:
            self.currency_value += self.valid_currency[currency]

        # Adds invalid currencies to a list to return it later
        else:
            self.invalid_currencies.append(currency)

    def return_invalid_currency(self):
        response = "Returning your invalid currency of "

        for i in range(len(self.invalid_currencies)):
            response += self.invalid_currencies[i]
            if i < len(self.invalid_currencies) - 1:
                response += ", "

        self.invalid_currencies.clear()

        return response

    def dispense_red(self):
        response = ""

        if self.invalid_currencies:
            response = self.return_invalid_currency()
        else:
            if self.currency_value >= 5:
                self.currency_value -= 5
                response = "Enjoy your red gumball"
            else:
                response = "You need at least 5 cents to dispense a red gumball"

        return response

    def dispense_yellow(self):
        response = ""

        if self.invalid_currencies:
            response = self.return_invalid_currency()
        else:
            if self.currency_value >= 10:
                self.currency_value -= 10
                response = "Enjoy your yellow gumball"
            else:
                response = "You need at least 10 cents to dispense a yellow gumball"

        return response
```

```python
    def return_my_change(self):
        response = ""

        if self.invalid_currencies:
            response = self.return_invalid_currency()
        else:
            if self.currency_value == 0:
                response = "There is no change to return"
            # Returns the user's change
            else:
                response = f"Returning your change of {self.currency_value} cents"
                self.currency_value = 0

        return response
```

`test_gumball_machine.py`

```python
import unittest
import gumball_machine as gumball_machine_class


class NoCurrencyTestCases(unittest.TestCase):
    def setUp(self):
        self.gumball_machine = gumball_machine_class.GumballMachine()

    def test_dispense_red(self):
        """Tests dispensing a red gumball with no currency in the machine"""
        # input: dispense_red()
        expected_output = "You need at least 5 cents to dispense a red gumball"

        actual_output = self.gumball_machine.dispense_red()
        self.assertEqual(expected_output, actual_output)

    def test_dispense_yellow(self):
        """Tests dispensing a yellow gumball with no currency in the machine"""
        # input: dispense_yellow()
        expected_output = "You need at least 10 cents to dispense a yellow gumball"

        actual_output = self.gumball_machine.dispense_yellow()
        self.assertEqual(expected_output, actual_output)

    def test_return_my_change(self):
        """Tests returning change with no currency in the machine"""
        # input: return_my_change()
        expected_output = "There is no change to return"

        actual_output = self.gumball_machine.return_my_change()
        self.assertEqual(expected_output, actual_output)


class ReturnValidCurrencyTestCases(unittest.TestCase):
    def setUp(self):
        self.gumball_machine = gumball_machine_class.GumballMachine()

    def test_return_nickel(self):
        """Tests inserting a nickel, then returning change"""
        # input: insert("nickel"), return_my_change()
        expected_output = "Returning your change of 5 cents"

        self.gumball_machine.insert("nickel")
        actual_output = self.gumball_machine.return_my_change()
        self.assertEqual(expected_output, actual_output)

    def test_return_dime(self):
        """Tests inserting a dime, then returning change"""
        # input: insert("dime"), return_my_change()
        expected_output = "Returning your change of 10 cents"

        self.gumball_machine.insert("dime")
        actual_output = self.gumball_machine.return_my_change()
        self.assertEqual(expected_output, actual_output)

    def test_return_quarter(self):
        """Tests inserting a quarter, then returning change"""
        # input: insert("quarter"), return_my_change()
        expected_output = "Returning your change of 25 cents"

        self.gumball_machine.insert("quarter")
        actual_output = self.gumball_machine.return_my_change()
        self.assertEqual(expected_output, actual_output)


class ExactCurrencyTestCases(unittest.TestCase):
    def setUp(self):
        self.gumball_machine = gumball_machine_class.GumballMachine()

    def test_insert_nickel_dispense_red(self):
        """Tests dispensing a red gumball with a nickel (5 cents) in the machine"""
        # input: insert("nickel"), dispense_red()
        expected_output = "Enjoy your red gumball"

        self.gumball_machine.insert("nickel")
        actual_output = self.gumball_machine.dispense_red()
        self.assertEqual(expected_output, actual_output)

    def test_insert_dime_dispense_yellow(self):
        """Tests dispensing a yellow gumball with a dime (10 cents) in the machine"""
        # input: insert("dime"), dispense_yellow()
        expected_output = "Enjoy your yellow gumball"

        self.gumball_machine.insert("dime")
        actual_output = self.gumball_machine.dispense_yellow()
        self.assertEqual(expected_output, actual_output)

    def test_insert_nickels_dispense_yellow(self):
        """Tests dispensing a yellow gumball with two nickels (10 cents) in the machine"""
        # input: insert("nickel"), insert("nickel"), dispense_yellow()
        expected_output = "Enjoy your yellow gumball"

        self.gumball_machine.insert("nickel")
        self.gumball_machine.insert("nickel")
        actual_output = self.gumball_machine.dispense_yellow()
        self.assertEqual(expected_output, actual_output)


class InvalidCurrencyTestCases(unittest.TestCase):
    def setUp(self):
        self.gumball_machine = gumball_machine_class.GumballMachine()

    def test_return_dollar(self):
        """Tests inserting a dollar, then returning change"""
        # input: insert("dollar"), return_my_change()
        expected_output = "Returning your invalid currency of dollar"

        self.gumball_machine.insert("dollar")
        actual_output = self.gumball_machine.return_my_change()
        self.assertEqual(expected_output, actual_output)

    def test_return_dollars(self):
        """Tests inserting multiple dollars, then returning change"""
        # input: insert("dollar"), insert("dollar"), return_my_change()
        expected_output = "Returning your invalid currency of dollar, dollar"
        self.gumball_machine.insert("dollar")
        self.gumball_machine.insert("dollar")
        actual_output = self.gumball_machine.return_my_change()
        self.assertEqual(expected_output, actual_output)

    def test_insert_dollar_dispense_red(self):
        """Tests inserting a dollar, then attempting to dispense a red gumball"""
        # input: insert("dollar"), dispense_red()
        expected_output = "Returning your invalid currency of dollar"

        self.gumball_machine.insert("dollar")
        actual_output = self.gumball_machine.dispense_red()
        self.assertEqual(expected_output, actual_output)

    def test_insert_dollar_dispense_yellow(self):
        """Tests inserting a dollar, then attempting to dispense a yellow gumball"""
        # input: insert("dollar"), yellow()
        expected_output = "Returning your invalid currency of dollar"

        self.gumball_machine.insert("dollar")
        actual_output = self.gumball_machine.dispense_yellow()
        self.assertEqual(expected_output, actual_output)


class MultipleGumballsExactCurrencyTestCases(unittest.TestCase):
    def setUp(self):
        self.gumball_machine = gumball_machine_class.GumballMachine()

    def test_insert_nickels_dispense_reds(self):
        """Tests inserting 2 nickels, and dispensing 2 red gumballs"""
        # input: insert("nickel"), insert("nickel"), dispense_red(), dispense_red(), return_my_change()
        expected_output = "There is no change to return"

        self.gumball_machine.insert("nickel")
        self.gumball_machine.insert("nickel")
        self.gumball_machine.dispense_red()
        self.gumball_machine.dispense_red()
        actual_output = self.gumball_machine.return_my_change()
        self.assertEqual(expected_output, actual_output)

    def test_insert_dime_dispense_reds(self):
        """Tests inserting a dime, and dispensing 2 red gumballs"""
        # input: insert("dime"), dispense_red(), dispense_red(), return_my_change()
        expected_output = "There is no change to return"

        self.gumball_machine.insert("dime")
        self.gumball_machine.dispense_red()
        self.gumball_machine.dispense_red()
        actual_output = self.gumball_machine.return_my_change()
        self.assertEqual(expected_output, actual_output)

    def test_insert_nickels_dispense_yellows(self):
        """Tests inserting 4 nickels, and dispensing 2 yellow gumballs"""
        # input: insert("nickel"), insert("nickel"), insert("nickel"), insert("nickel"), dispense_yellow(), dispense_y
        expected_output = "There is no change to return"

        self.gumball_machine.insert("nickel")
        self.gumball_machine.insert("nickel")
        self.gumball_machine.insert("nickel")
        self.gumball_machine.insert("nickel")
        self.gumball_machine.dispense_yellow()
        self.gumball_machine.dispense_yellow()
        actual_output = self.gumball_machine.return_my_change()
        self.assertEqual(expected_output, actual_output)

    def test_insert_dimes_dispense_yellows(self):
        """Tests inserting 2 dimes, and dispensing 2 yellow gumballs"""
        # input: insert("dime"), insert("dime"), dispense_yellow(), dispense_yellow(), return_my_change()
        expected_output = "There is no change to return"

        self.gumball_machine.insert("dime")
        self.gumball_machine.insert("dime")
        self.gumball_machine.dispense_yellow()
        self.gumball_machine.dispense_yellow()
        actual_output = self.gumball_machine.return_my_change()
        self.assertEqual(expected_output, actual_output)

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

Test Cases

NoCurrencyTestCases

These test cases focus on inserting no currency into the machine and attempting to dispense gumballs and returning change.

Test Name	Input Vector	Expected Output
test_dispense_red	< `dispense_red()` >	"You need at least 5 cents to dispense a red gumball"
test_dispense_yellow	< `dispense_yellow()` >	"You need at least 10 cents to dispense a yellow gumball"
test_return_my_change	< `return_my_change()` >	"There is no change to return"

ReturnValidCurrencyTestCases

These test cases focus on inserting valid currency into the machine and returning change.

Test Name	Input Vector	Expected Output
test_return_nickel	< `insert("nickel")` , `return_my_change()` >	"Returning your change of 5 cents"
test_return_dime	< `insert("dime")` , `return_my_change()` >	"Returning your change of 10 cents"
test_return_quarter	< `insert("quarter")` , `return_my_change()` >	"Returning your change of 25 cents"

ExactCurrencyTestCases

These test cases focus on inserting exact, valid currency into the machine and dispensing gumballs.

Test Name	Input Vector	Expected Output
test_insert_nickel_dispense_red	< `insert("nickel")` , `dispense_red()` >	"Enjoy your red gumball"
test_insert_dime_dispense_yellow	< `insert("nickel")` , `dispense_yellow()` >	"Enjoy your yellow gumball"
test_insert_nickels_dispense_yellow	< `insert("nickel")` , `insert("nickel")` , `dispense_yellow()` >	"Enjoy your yellow gumball"

InvalidCurrencyTestCases

These test cases focus on inserting invalid currency into the machine and attempting to dispense gumballs and returning change.

Test Name	Input Vector	Expected Output
test_return_dollar	< `insert("dollar")` , `return_my_change()` >	"Returning your invalid currency of dollar"

Test Name	Input Vector	Expected Output
test_return_dollars	< `insert("dollar")`, `insert("dollar")`, `return_my_change()` >	"Returning your invalid currency of dollar, dollar"
test_insert_dollar_dispense_red	< `insert("dollar")`, `dispense_red()` >	"Returning your invalid currency of dollar"
test_insert_dollar_dispense_yellow	< `insert("dollar")`, `dispense_yellow()` >	"Returning your invalid currency of dollar"

MultipleGumballsExactCurrencyTestCases

These test cases focus on inserting exact, valid currency multiple gumballs.

Test Name	Input Vector	Expected Output
test_insert_nickels_dispense_reds	< `insert("nickel")`, `insert("nickel")`, `dispense_red()`, `dispense_red()`, `return_my_change()` >	"There is no change to return"
test_insert_dime_dispense_reds	< `insert("dime")`, `dispense_red()`, `dispense_red()`, `return_my_change()` >	"There is no change to return"
test_insert_nickels_dispense_yellows	< `insert("nickel")`, `insert("nickel")`, `insert("nickel")`, `insert("nickel")`, `dispense_yellow()`, `dispense_yellow()`, `return_my_change()` >	"There is no change to return"
test_insert_dimes_dispense_yellows	< `insert("dime")`, `insert("dime")`, `dispense_yellow()`, `dispense_yellow()`, `return_my_change()` >	"There is no change to return"