```
# DWAYNE FRASER
# PROBLEM 1

def testif(b, testname, msgOK="", msgFailed=""):
    """Function used for testing.
    param b: boolean, normally a tested condition: true if test passed, false otherwise
    param testname: the test name
    param msgOK: string to be printed if param b==True  ( test condition true)
    param msgFailed: string to be printed if param b==False
    returns b
    """
    if b:
        print("Success: "+ testname + "; " + msgOK)
    else:
        print("Failed: "+ testname + "; " + msgFailed)
    return b

class NVector: #Defines the Class
    """ N Vector Class
    """
    # INITIALIZATION
    def __init__(self, *content): # CONSTRUCTOR
        """ Constructor
        """
        try:
            if len(content) == 1:
                self.state = list(content[0])
            else:
                self.state = [] # creates vector
                for i in content: # loop
                    self.state.append(i) # append
        except IndexError:
            raise

    def __len__(self):
        """ Returns Length
        """
        return len(self.state)


    def __getitem__(self, index):
        """ Gets Index
        """
        return self.state[index] # FINDS INDEX

    def __setitem__(self, index, value):
        """ Assigns Value
        """
        self.state[index] = value # SETS VALUE

    def __str__(self):
        """ String Representation
```

```
        """
        string_varible = str(self.state)
        return string_varible # RETURNS VECTOR AS A STRING

    def __eq__(self, other):
        """ Equals Function
        """
        return (self.state == other.state) # BOOLEAN

    def __ne__(self, other):
        """ Not Equal Function
        """
        return (self.state != other.state) # BOOLEAN

    def __add__(self, other):
        """ Addition Function
        """
        try:
            temp = []
            for i in range(0, len(self.state)): # LOOP
                temp.append(self.state[i] + other.state[i]) # ADD
            return temp
        except IndexError:
            raise

    def __radd__(self, other):
        """ Reflected Addition
        """
        try:
            temp = []
            for i in range(0, len(self.state)): # LOOP
                temp.append(other.state[i] + self.state[i]) # R ADD
            return temp
        except IndexError:
            raise

    def __mul__(self, other):
        """ Multiplication
        """
        try:
            temp = []
            for i in range(0, len(self.state)): # LOOP
                temp.append(self.state[i] * other.state[i]) # MUL
            return temp
        except IndexError:
            raise

    def __rmul__(self, other):
        """ Reflected Multiplication
        """
        try:
            temp = []
```

```
            for i in range(0, len(self.state)): # LOOP
                temp.append(other.state[i] * self.state[i]) # R MUL
            print(temp)
            return temp
        except IndexError:
            raise

    def __zero__(self, n):
        """ Zeros Function
        """
        self.state = []
        for i in range(0, n): # LOOP
            self.state.append(0)
        print(self.state)
        return self.state # Returns N Zeros

def main():

    print("TEST FUNCTIONS")
    vector = NVector([1,2,3,4])
    print(vector)
    print("Testing Constructor")
    testif(vector.state == [1,2,3,4], "Initilization", "Successful", "Failed")

    print("\nTesting Length Function")
    testif(vector.__len__() == 4, "Length Function", "Successful", "Failed")

    print("\nTesting Get Item Function")
    testif(vector.__getitem__(0) == 1, "Get Item Function", "Successful", "Failed")

    print("\nSet Item Function")
    vector.__setitem__(3,5)
    testif(vector.__getitem__(3) == 5, "Set Item Function", "Successful", "Failed")

    print("\nString Representation Function")
    vector = NVector(0,1,2,3,4,5)
    print(vector)
    testif(vector.__str__() == "[0, 1, 2, 3, 4, 5]", "String Representation Function", "Successful", "Failed")

    print("\nEQUAL/NOT EQUAL FUNCTIONS")
    vector2 = NVector(0,1,2,3,4,5)
    vector3 = NVector(5,4,3,2,1,0)
    testif(vector.__eq__(vector2), "EQ Function", "Successful", "Failed")
    testif(vector.__ne__(vector3), "Not EQ Function", "Successful", "Failed")

    print("\nAdd Function")
    new_vector = NVector(1,1,1,1,1,1)
    testif(vector.__add__(new_vector) == [1,2,3,4,5,6], "Add Function", "Successful", "Failed")

    print("\nR-Add Function")
    testif(vector.__radd__(new_vector) == [1,2,3,4,5,6], "R-Add Function", "Successful", "Failed")
```

```
    print("\nMUL Function")
    new_vector = NVector(10,10,10,10,10,10)
    testif(vector.__mul__(new_vector) == [0,10,20,30,40,50], "MULTIPLY Function", "Successful", "Failed")

    print("\nR-MUL Function")
    testif(vector.__rmul__(new_vector) == [0,10,20,30,40,50], "R-MULTIPLY Function", "Successful", "Failed")

    print("\nZeros Function")
    vector = NVector(0,0,0,0)
    testif(vector.__zero__(5) == [0, 0, 0, 0, 0], "Zeros Function", "Successful", "Failed")

main()
```

```
In [1]: runfile('C:/Program Files (x86)/Work/Python/Python Dwayne Solutions/HW 3/p1_Fraser_Dwayne.py', wdir='C:/Program Files (x86)/Work/Python/Python
Dwayne Solutions/HW 3')
TEST FUNCTIONS
[1, 2, 3, 4]
Testing Constructor
Success: Initilization; Successful

Testing Length Function
Success: Length Function; Successful

Testing Get Item Function
Success: Get Item Function; Successful

Set Item Function
Success: Set Item Function; Successful

String Representation Function
[0, 1, 2, 3, 4, 5]
Success: String Representation Function; Successful

EQUAL/NOT EQUAL FUNCTIONS
Success: EQ Function; Successful
Success: Not EQ Function; Successful

Add Function
Success: Add Function; Successful

R-Add Function
Success: R-Add Function; Successful

MUL Function
Success: MULTIPLY Function; Successful

R-MUL Function
[0, 10, 20, 30, 40, 50]
Success: R-MULTIPLY Function; Successful

Zeros Function
[0, 0, 0, 0, 0]
Success: Zeros Function; Successful

In [2]:
```