```
"""
IPv4 Conversion
"""
__author__ = "John Priest"
__version__ = "Fall 2023"

def main():
    """
    Prompts the user for an IPv4 address, converts it to binary, and prints the
result.
    """
     address = input("Enter an IPv4 address: ")
    binary_address = convert_to_binary(address)
    print(binary_address)

def convert_to_binary(address):
    """
    Converts an IPv4 address from dotted decimal to binary
    Args:
        address a string is in dotted decimal notation
    Returns:
        dotted decimal notation of 8-bit binary number of IPv4 address.
    """
    groups = address.split(".")
    if len(groups) != 4:
        return "Invalid IP address.
    octets = []
    answer = ""

    for group in groups:
        if not group.isdigit():
            return "Invalid IP address."
        group = int(group)

        if not (0 <= group <= 255):
            return "Invalid IP address."

        binary_group = bin(group)[2:]

        if len(binary_group) < 8:
            binary_group = pad(binary_group)

        octets.append(binary_group)

    answer = ".".join(octets)
    return answer

def pad(binary_group):
    """
    Private helper function to pad a binary number with leading 0s.
    Args:
        binary_group: Binary number to be padded.
    Returns:
        Padded binary number.
    """
     octet = ""
    pad_size = 8 - len(binary_group)
    while pad_size > 0:
        octet = '0' + octet
        pad_size -= 1
    octet = octet + binary_group
return octet

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

```python
"""
Tests for ipv_john_priest.py
Ensures correct functionality of ipv_john_priest.py
"""
import ipv4_john_priest as jp
import unittest
__author__ = "John Priest"
__version__ = "Fall 2023"

class TestConvertToBinary(unittest.TestCase):
     def test_valid_ip(self):
        address = '126.255.255.254'
        expected = '01111110.11111111.11111111.11111110'
        result = jp.convert_to_binary(address)
        self.assertEqual(expected, result)

    def test_valid_zero_ip(self):
        address = '0.0.0.0'
        expected = '00000000.00000000.00000000.00000000'
        result = jp.convert_to_binary(address)
        self.assertEqual(expected, result)

    def test_valid_repeating_ip(self):
        address = '255.255.255.255'
        expected = '11111111.11111111.11111111.11111111'
        result = jp.convert_to_binary(address)
        self.assertEqual(expected, result)

    def test_invalid_ip_length_short(self):
        address = '192.168.1'
        expected = 'Invalid IP address.'
        result = jp.convert_to_binary(address)
        self.assertEqual(expected, result)

    def test_invalid_ip_length_long(self):
        address = '254.168.1.1.1'
        expected = 'Invalid IP address.'
        result = jp.convert_to_binary(address)
        self.assertEqual(expected, result)

     def test_invalid_ip_empty(self):
        address = ' . . . '
        expected = 'Invalid IP address.'
        result = jp.convert_to_binary(address)
        self.assertEqual(expected, result)

     def test_invalid_ip_nonnumeric(self):
        address = 'abc.168.1.1'
        expected = 'Invalid IP address.'
        result = jp.convert_to_binary(address)
        self.assertEqual(expected, result)

    def test_binary_padding(self):
        address = '10.5.68.1'
        expected = '00001010.00000101.01000100.00000001'
        result = jp.convert_to_binary(address)
        self.assertEqual(expected, result)

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