bitstring_class

February 13, 2026

1 BitString Class

Write a Class that implements a bit representation that provides the functionality requested in the following questions.

[155]:

```python
import numpy as np
import math
import copy as cp


class BitString:
    """
    Simple class to implement a config of bits
    """
    def __init__(self, N):
        self.N = N
        self.config = np.zeros(N, dtype=int)

    def __repr__(self):
        out = ""
        for i in self.config:
            out += str(i)
        return out

    def __eq__(self, other):
        return all(self.config == other.config)

    def __len__(self):
        return len(self.config)

    def on(self):
        """
        Return number of bits that are on
        """
        return np.sum(self.config)

    def off(self):
        """
        Return number of bits that are off
        """
        return self.N - self.on()

    def flip_site(self, i):
        """
        Flip the bit at site i
        """
        self.config[i] = 1 - self.config[i]

    def integer(self):
        """
        Return the decimal integer corresponding to BitString
        """
        val = 0
        for i in range(self.N):
            val += self.config[i] * 2**(self.N - 1 - i)
        return val

    def set_config(self, s:list[int]):
        """
        Set the config from a list of integers
        """
        self.config = np.array(s, dtype=int)

    def set_integer_config(self, dec:int):
        """
        convert a decimal integer to binary

        Parameters
        ----------
        dec    : int
            input integer

        Returns
        -------
        Bitconfig
        """
        self.config = np.zeros(self.N, dtype=int)
        for i in range(self.N):
            self.config[self.N - 1 - i] = dec % 2
            dec = dec // 2
```

---

1. Create an zero `BitString` of length 8 and flip a few bits and print the output.

Methods needed: - `__str__()` - `flip()` - `__len__()`

[156]:

```
my_bs = BitString(8)
my_bs.flip_site(2)
my_bs.flip_site(2)
print(" The following should be 0:")
print(my_bs)

my_bs.flip_site(2)
my_bs.flip_site(7)
my_bs.flip_site(0)
print(" The following should have 0,2,7 bits flipped:")
print(my_bs)

print(" Length of bitstring: ", len(my_bs))
assert(len(my_bs) == 8)
```

```
 The following should be 0:
00000000
 The following should have 0,2,7 bits flipped:
10100001
 Length of bitstring:  8
```

---

2. Add a method that lets you directly set the value of the bitstring by providing a string of 0s and 1s:

Methods needed: - `set_config()`

[157]:

```
my_bs = BitString(13)
my_bs.set_config([0,1,1,0,0,1,0,0,1,0,1,0,0])
print(my_bs)
```

```
0110010010100
```

---

3. Add a method that returns number of `on` bits and one that returns the number of `off` bits.

Methods needed: - `on()` - `off()`

[158]:

```
print(" on:  ", my_bs.on())
print(" off: ", my_bs.off())
assert(my_bs.on() == 5)
assert(my_bs.off() == 8)
```

```
 on:   5
 off:  8
```

---

4. Add a method that returns the associated integer (decimal).

Methods needed: - `integer()`

[159]:

```python
print(my_bs.integer())
assert(my_bs.integer() == 3220)
```

```
3220
```

---

5. Add a method that lets you directly set the value of the bitstring by providing a decimal integer.

Also include an optional keyword `digits` to let the user specify the length of the string.

Methods needed: - `set_integer_config()`

[160]:

```python
my_bs = BitString(20)
my_bs.set_integer_config(3221)
print(my_bs)

# Let's make sure this worked:
tmp = np.array([0,0,0,0,0,0,0,0,1,1,0,0,1,0,0,1,0,1,0,1])
assert((my_bs.config == tmp).all())

# We can provide an even stronger test here:
for i in range(1000):
    my_bs.set_integer_config(i) # Converts from integer to binary
    assert(my_bs.integer() == i) # Converts back from binary to integer and tests
```

```
00000000110010010101
```

---

6. Overload equality operator

Methods needed: - `__eq__()`

[161]:

```python
my_bs1 = BitString(13)
my_bs1.set_config([0,1,1,0,0,1,0,1,1,0,1,0,0])
print(my_bs1, ": ", my_bs1.integer())

my_bs2 = BitString(13)
my_bs2.set_integer_config(3252)
print(my_bs2, ": ", my_bs2.integer())


assert(my_bs1 == my_bs2)

my_bs2.flip_site(5)
assert(my_bs1 != my_bs2)
```

```
0110010110100 :  3252
0110010110100 :  3252
```