You will find a private GitHub repo called <github-username>-hw where you will submit all your homework assignments. All your code should appear in a file called hw1.py that lives inside the hw1 directory. Make sure to commit your changes with $ git commit -a -m "log message about hw 1" and push them with $ git push. No need to make new branches.

**Question 1 (10 points).** Let \( l = \text{list('diving into the deluge of data')} \). Without using the python interpreter, but with the use of documentation, what does \( ''.join(l) \) equal after performing the following operations? Verify your answer on the computer. Were you right? If not, where did you go wrong? Give your guess and explanation in a comment (i.e., a line starting with #) in hw1.py.

```python
>>> l.remove('i')
>>> del l[1]
>>> del l[4:9]
>>> l.reverse()
>>> del l[:8]
>>> l.reverse()
>>> l.pop()
>>> l.append('a')
>>> l[-6] = 'b'
```

**Question 2 (10 points).** Write a function called `star_range(n)` that, given \( n \) prints all the numbers from 1 to \( n \) delimited by an asterisk. For example:

```python
>>> star_range(5)
"1*2*3*4*5"
>>> star_range(1)
"1"
```

**Question 3.** In class we wrote code to convert a number in decimal (i.e., base 10) into binary. Here you will write a function that converts a binary number into a decimal one. For example:

```python
>>> convert_to_decimal([1,0,1,1,1])
23
>>> convert_to_decimal([1])
1
>>> convert_to_decimal([1,1])
3
```

Imagine you were only considering the first three bits of the binary number 10111. These bits are 101. This number in decimal is 5 because we have

\[
5 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0.
\]

Now imagine that you want to consider the first four bits——1011—which has the polynomial expansion

\[
11 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 2 \times (1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0) + 1 \times 2^0.
\]

In other words, if we know that the first three bits yield 5 in decimal, then the first four bits yield twice the value of the first three bits plus the value of the new bit (either 0 or 1). This provides a very succinct algorithm for converting a sequence of bits \( b \) into a decimal number—if \( \text{val} \) corresponds to the decimal value of the first \( i \) bits (i.e., \( b[:i] \)) then \( 2 \times \text{val} + b[i] \) is the decimal value of the first \( (i+1) \) bits. Notice that a good starting value is 0.