Look-and-Say Sequence

In this lesson, you will learn how to generate the next term of the Look-and-Say sequence in Python.

We'll cover the following

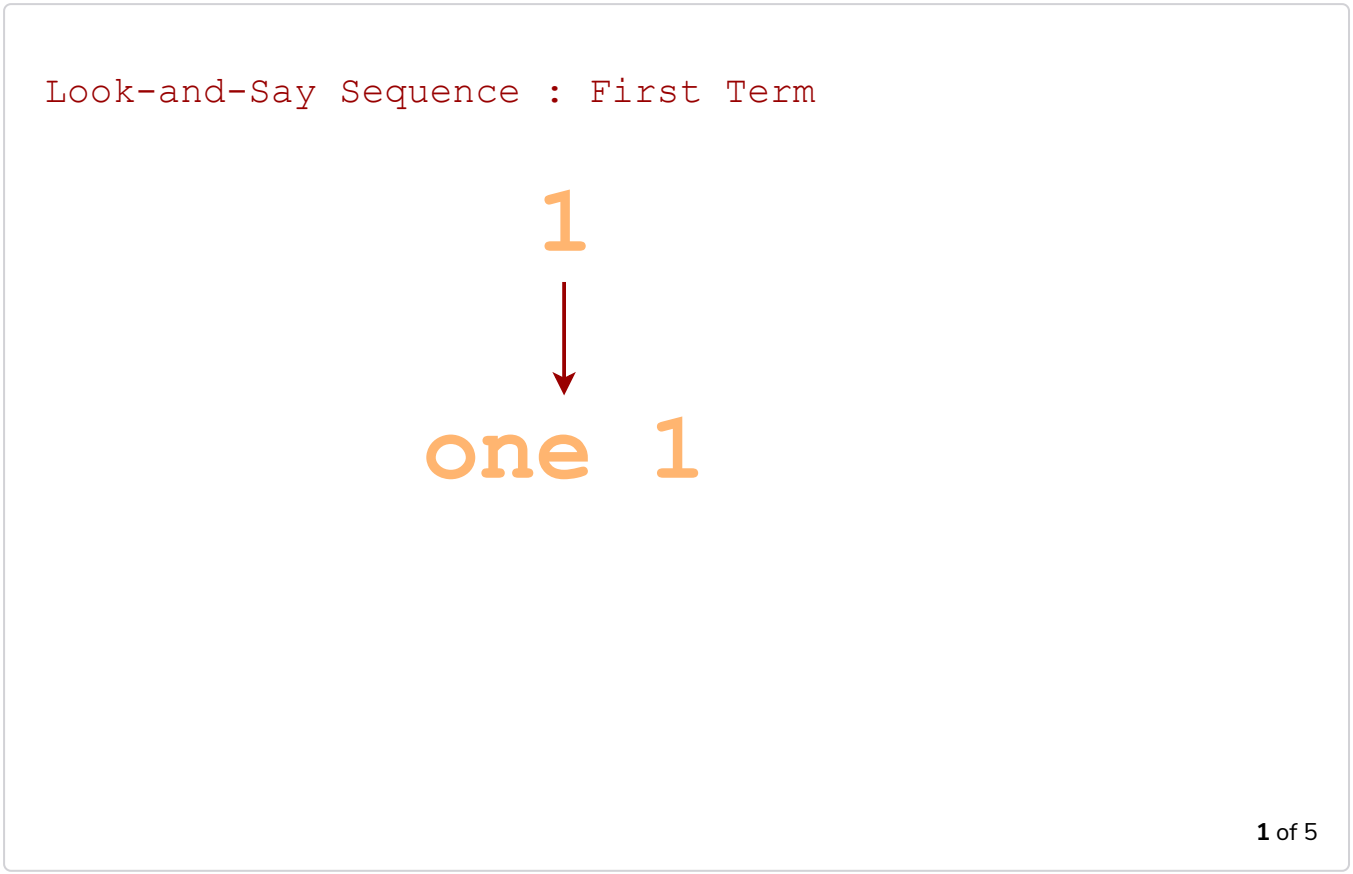
In this lesson, we will be considering the so-called “Look-and-Say” sequence. The first few terms of the sequence are:

```
1, 11, 21, 1211, 111221, 312211, 13112221, 1113213211, ...
```

To generate a member of the sequence from the previous member, read off the digits of the previous member and record the count of the number of digits in groups of the same digit.

For example, `1` is read off as one `1` which implies that the count of `1` is one. As `1` is read off as “one 1”, the next sequence will be written as `11` where `1` replaces `one`. Now `11` is read off as “two 1s” as the count of “1” is two in this term. Hence, the next term in the sequence becomes `21`.

Have a look at the slides below where we generate this sequence up to the fifth term.



Now, you can easily guess the sixth term. There you go:

```
111221 is read off as "three 1s, two 2s, then one 1" or 312211.
```

Implementation

Hopefully, by now, you understand the look-and-say sequence. Let's have a look at the implementation in Python below:

```python
def next_number(s):
    result = []
    i = 0
    while i < len(s):
        count = 1
        while i + 1 < len(s) and s[i] == s[i+1]:
            i += 1
            count += 1
        result.append(str(count) + s[i])
        i += 1
    return ''.join(result)
```

next_number(s)

Explanation

In the `next_number` function, `result` is initialized to an empty list on **line 2**. `i` is set to `0` in the next line so that we can traverse the string, `s`, from the first character in the `while` loop on **line 4** which runs as long as `i` is less than the length of `s`. On **line 5**, in the outer `while` loop, `count` is set to `1` before the execution proceeds to the inner `while` loop. The inner `while` loop on **line 6** runs until the value of `i` does not exceed the length of `s` and the current and next character of `s` indicated by `i` are the same. So essentially, in the inner `while` loop, we are keeping a count of consecutive similar characters as we increment `i` and `count` by `1` in each iteration of the inner `while` loop (**lines 7-8**).

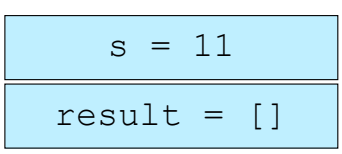
As soon as the inner `while` loop terminates due to change of character in the string, or the code has reached the end of the string, the execution jumps to **line 9**. On **line 9**, `count` is converted to a string and `s[i]`, which is the number we have the `count` for, is concatenated and appended to `result`. `i` is incremented on **line 10** to iterate to the next character in the next iteration of the outer `while` loop where `count` resets to `1` once again to count for the next number.

After `s` is traversed entirely, all the elements in `result` are concatenated using the `join` function, which joins all the elements in `result` without any space as specified by the `''` separator. Then, a single string is returned from the function `next_number`.

Let's visualize the execution with the help of the slides below:

Look-and-Say Sequence

```
  def next_number(s):
      result = []
      i = 0
      while i < len(s):
          count = 1
          while i + 1 < len(s) and s[i] == s[i+1]:
              i += 1
              count += 1
          result.append(str(count) + s[i])
          i += 1
      return ''.join(result)
```



In the code below, by using a `for` loop on **line 16**, we generate the next term of the look-and-say sequence from the previous term and print out a total of four terms.

```
def next_number(s):
    result = []
    i = 0
    while i < len(s):
        count = 1
        while i + 1 < len(s) and s[i] == s[i+1]:
            i += 1
            count += 1
        result.append(str(count) + s[i])
        i += 1
    return ''.join(result)

s = "1"
print(s)
n = 4
for i in range(n-1):
    s = next_number(s)
    print(s)
```

I hope you were able to understand the string processing regarding the look-and-say sequence that we performed in this lesson. Stay tuned for more problems regarding string processing in the next few lessons.