1 Range function

The function `range` is another builtin function. It takes one argument and returns a sequence of numbers. The length of the returned sequence is equal to the value of the argument: `range(4)` returns a sequence of 4 numbers. In general, for some number `n`, the function call `range(n)` returns the sequence `0, ..., n-1`. The sequence is something called a list. We will learn about lists later. For now, our only use of `range` will be inside for loops.

```
>>> range(4)
[0, 1, 2, 3]
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(0)   # returns an empty sequence
[]
```

2 For loops using range

The last handout described the basics of the `for` statement. It showed how to loop over the characters of a string. Recall that a string is a *sequence* of characters. The output of `range` is a *sequence* of numbers. It turns out that we can use a `for` statement to loop over *any sequence*. This loop prints the numbers 0 to 9, each on a separate line.

```
for num in range(10):
    print num
```

3 Making complex patterns with for loops

The `for` statement is a powerful tool with many applications. A `for` loop repeats the same body of code over and over: the challenge is to figure out how to express what you want to do as repetitive application of some pattern. Here we look at a fun application: text art. We can use the `for` statement to write a program that draws a "vee" shape like this:

```
.........
 .......
  .....
   ...
    .
```

The pattern is actually made up of *two* characters: not only dots, but also spaces. The number of each varies by line. We can use a `for` loop to print this pattern: each time through the loop we will print one line of text. The challenge is figuring out how many dots and spaces to print on each line.

```
for line in range(5):
    spaces =  # how many??
    dots =    # how many??
    print ' ' * spaces + '.' * dots
```

The pattern for spaces is pretty easy: there are 0 on the first line, 1 on the second, 2 on the third, and so on. Therefore, the number of spaces is equal to the value of the loop variable `line`. To help us figure out the pattern for the dots, we can use a **loop table**.

line	dots	`-2 * line`	`-2 * line + 9`
0	9	0	0 + 9
1	7	-2	-2 + 9
2	5	-4	-4 + 9
3	3	-6	-6 + 9
4	1	-8	-8 + 9

The first thing to is figure out the rate of change: each time the `line` goes up by 1, the number of dots goes down by 2. The expression `-2 * line` has the desired rate of change but the numbers are off. In fact, every number is off by exactly 9. To complete the formula, we can add 9 to the expression: `-2 * line + 9` yields the correct number of dots for every line.

```
for line in range(5):
    spaces = line
    dots = -2 * line + 9
    print ' ' * spaces + '.' * dots
```

To make this more interesting, allow the size of the vee to vary. Here is size 3:

```
.....
 ...
  .
```

Here is size 4:

```
.......
 .....
  ...
   .
```

The size affects the pattern in two ways: the number of lines, and the number of dots on the first line. To figure out how the number of dots changes as a function of size, we again use a loop table.

size	dots on first row	`2 * size`	`2 * size - 1`
3	5	6	6 - 1
4	7	8	8 - 1
5	9	10	10 - 1

```
size = int(raw_input("Enter a size (1 or larger): "))
for line in range(size):                  # loop repeats size times
    spaces = line
    dots = -2 * line + (2 * size - 1)     # number of dots depends on size
    print ' ' * spaces + '.' * dots
```

4 Exercises

1. Write a short program that produces a scalable staircase. It should work like this:

```
Enter a size (1 or larger): 3
Staircase:
----//..\\----
--//......\\--
//..........\\
```

This time the user enters 6.

```
Enter a size (1 or larger): 6
Staircase:
----------//..\\----------
--------//......\\--------
------//..........\\------
----//..............\\----
--//..................\\--
//......................\\
```

Solution:

```
size = int(raw_input("Enter a size (1 or larger): "))
print "Staircase:"

for line in range(size):
    dashes = -2*line + (2*size-2)
    dots = 4*line + 2
    print '-'*dashes + '//' + '.'*dots + '\\\\' + '-'*dashes
```