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.

```python
>>> 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.

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

Examples

```python
for i in range(10):
    squared = i * i
    print(i, '^2 =', squared)
    print(i)
```

```python
# This program will average numbers given by the user.
how_many_numbers = eval(input("How many numbers do you want to average? "))
sum = 0.0  # initialize accumulator variable

for x in range(how_many_numbers):
    userNum = eval(input("Enter a number: "))
    sum = sum + userNum  # add current number

average = sum/how_many_numbers
print("The average is", average)
```
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.

<table>
<thead>
<tr>
<th>line</th>
<th>dots</th>
<th>-2 * line</th>
<th>-2 * line + 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>9</td>
<td>0</td>
<td>0 + 9</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>-2</td>
<td>-2 + 9</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>-4</td>
<td>-4 + 9</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>-6</td>
<td>-6 + 9</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>-8</td>
<td>-8 + 9</td>
</tr>
</tbody>
</table>

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 $-2 \times \text{line}$ 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: $-2 \times \text{line} + 9$ 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.

<table>
<thead>
<tr>
<th>size</th>
<th>dots on first row</th>
<th>2 * size</th>
<th>2 * size - 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>5</td>
<td>6</td>
<td>6 - 1</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>8</td>
<td>8 - 1</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>10</td>
<td>10 - 1</td>
</tr>
</tbody>
</table>

```
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  In class exercise

Enter a size (1 or larger): 4
Staircase:
------//..\------
----//......\----
--//..........\--
//..............\ 

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

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

//.........................\ 