codility
WE TEST CODERS

Chapter 1

Iterations

In programming, iterating means repeating some part of your program. This lesson presents basic programming constructions that allow iterations to be performed: "for" and "while" loops.

1.1. For loops

If you want to repeat some operations a given number of times, or repeat them for each element in some collection, a "for" loop is the right tool to use. Its syntax is as follows:

1.1: For loop syntax

```
1  for some_variable in range_of_values:
2      loop_body
```

The for loop repeats `loop_body` for each value in turn from the `range_of_values`, with the current value assigned to `some_variable`. In its simplest form, the range of values can be a range of integers, denoted by: `range(lowest, highest + 1)`. For example, the following loop prints every integer from 0 to 99:

```
1  for i in range(0, 100):
2      print i
```

Looping over a range of integers starting from 0 is a very common operation. (This is mainly because arrays and Python lists are indexed by integers starting from 0; see Chapter 2 Arrays for more details.) When specifying the range of integers, if the starting value equals zero then you can simply skip it. For example, the following loop produces exactly the same result as the previous one:

```
1  for i in range(100):
2      print i
```

Example: We are given some positive integer n. Let's compute the factorial of n and assign it to the variable `factorial`. The factorial of n is $n! = 1 \cdot 2 \cdot \ldots \cdot n$. We can obtain it by starting with 1 and multiplying it by all the integers from 1 to n.

```
1  factorial = 1
2  for i in range (1, n + 1):
3      factorial *= i
```

Example: Let's print a triangle made of asterisks ('$\star$') separated by spaces. The triangle should consist of n rows, where n is a given positive integer, and consecutive rows should contain 1, 2, ..., n asterisks. For example, for $n = 4$ the triangle should appear as follows:

```
*
* *
* * *
* * * *
```

We need to use two loops, one inside the other: the outer loop should print one row in each step and the inner loop should print one asterisk in each step[2].

```
1 for i in range(1, n + 1):
2     for j in range(i):
3         print '*',
4     print
```

The `range` function can also accept one more argument specifying the step with which the iterated values progress. More formally, `range(start, stop, step)` is a sequence of values beginning with `start`, whose every consecutive value is increased by `step`, and that contains only values smaller than `stop` (for positive `step`; or greater than `stop` for negative `step`). For example, `range(10, 0, -1)` represents sequence 10, 9, 8, ..., 1. Note that we cannot omit `start` when we specify `step`.

Example: Let's print a triangle made of asterisks ('$\star$') separated by spaces and consisting of n rows again, but this time upside down, and make it symmetrical. Consecutive rows should contain $2n - 1$, $2n - 3$, ..., 3, 1 asterisks and should be indented by 0, 2, 4, ..., $2(n - 1)$ spaces. For example, for $n = 4$ the triangle should appear as follows:

```
* * * * * * *
  * * * * *
    * * *
      *
```

The triangle should have n rows, where n is some given positive integer.

This time we will use three loops: one outer and two inner loops. The outer loop in each step prints one row of the triangle. The first inner loop is responsible for printing the indentations, and the second for printing the asterisks.

```
1 for i in range(n, 0, -1):
2     for j in range(n - i):
3         print ' ',
4     for j in range(2 * i - 1):
5         print '*',
6     print
```

1.2. While loops

The number of steps in a for loop, and the values over which we loop, are fixed before the loop starts. What if the number of steps is not known in advance, or the values over which

[2]There is a clever idiom in Python denoting a string repeated a number of times. For example, `'*' * n` denotes a string comprising n asterisks. But to make our examples more instructive, we will not use this idiom here. Instead we will use the `print` statement ended with a comma. It doesn't print the newline character, but follows the output with a single space.

we loop are generated one by one, and are thus not known in advance either? In such a case, we have to use a different kind of loop, called a "while" loop. The syntax of the while loop is as follows:

1.2: While loop syntax

```
while some_condition:
    loop_body
```

Before each step of the loop, `some_condition` is computed. As long as its value is true[3], the body of the loop is executed. Once it becomes false, we exit the loop without executing `loop_body`.

Example: Given a positive integer n, how can we count the number of digits in its decimal representation? One way to do it is convert the integer into a string and count the characters. Here, though, we will use only arithmetical operations instead. We can simply keep dividing the number by ten and count how many steps are needed to obtain 0.

```
result = 0
while n > 0:
    n = n // 10
    result += 1
```

Example: The Fibonacci numbers[4] form a sequence of integers defined recursively in the following way. The first two numbers in the Fibonacci sequence are 0 and 1, and each subsequent number is the sum of the previous two. The first few elements in this sequence are: 0, 1, 1, 2, 3, 5, 8, 13. Let's write a program that prints all the Fibonacci numbers, not exceeding a given integer n.

We can keep generating and printing consecutive Fibonacci numbers until we exceed n. In each step it's enough to store only two consecutive Fibonacci numbers.

```
a = 0
b = 1
while a <= n:
    print a
    c = a + b
    a = b
    b = c
```

1.3. Looping over collections of values[5]

We have seen how to loop over integers. Is it possible to loop over values of other types? Yes: using a "for" loop, we can loop over values stored in virtually any kind of container. The `range` function constructs a list containing all the values over which we should loop. However, we can pass a list constructed in any other way.

[3]Note that the condition can yield any value, not only `True` or `False`. A number of values other than `False` are interpreted as false, e.g. `None`, `0`, `[]` (empty list) and `''` (empty string).

[4]You can read more about the Fibonacci numbers in Chapter 13.

[5]If you are not familiar with various built-in data structures such as lists, sets and dictionaries, you can safely skip this section. Looping over lists of values will be explained in Chapter 2.

Example: The following program:

```
1 days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday',
2         'Friday', 'Saturday', 'Sunday']
3 for day in days:
4     print day
```

prints all the days of the week, one per line.

When we use the `range` function, we first build a list of all the integers over which we will loop; then we start looping. This is memory-consuming when looping over a long sequence. In such cases, it's advisable to use – instead of `range` – an equivalent function called `xrange`. This returns exactly the same sequence of integers, but instead of storing them in a list, it returns an object that generates them on the fly.

If you loop over a set of values, the body of the loop is executed exactly once for every value in the set; however, the order in which the values are processed is arbitrary.

Example: If we modify the above program slightly, as follows:

```
1 days = set(['Monday', 'Tuesday', 'Wednesday', 'Thursday',
2             'Friday', 'Saturday', 'Sunday'])
3 for day in days:
4     print day
```

we might get the days output in some strange order, e.g.:

```
Monday
Tuesday
Friday
Wednesday
Thursday
Sunday
Saturday
```

Looping over a dictionary means looping over its set of keys. Again, the order in which the keys are processed is arbitrary.

Example: The following program:

```
1 days = {'mon': 'Monday', 'tue': 'Tuesday', 'wed': 'Wednesday',
2         'thu': 'Thursday', 'fri': 'Friday', 'sat': 'Saturday',
3         'sun': 'Sunday'}
4 for day in days:
5     print day, 'stands for', days[day]
```

might output e.g.:

```
wed stands for Wednesday
sun stands for Sunday
fri stands for Friday
tue stands for Tuesday
mon stands for Monday
thu stands for Thursday
sat stands for Saturday
```

Every lesson will provide you with programming tasks at `http://codility.com/programmers`.