Unit 2.1: Loops

August 24, 2021

In the last unit we talked about boolean expressions and conditional branching. Boolean expressions allow us to express conditions and to check at runtime if they are `True` or `False`. With the `if` statement we can deviate from the normal sequential control-flow depending on whether a certain condition is true or not. This allows us to let our program behave differently in different situations.

In this unit we will deal with the implementation of repetitive behavior (loops). The lecture will cover loops with the `while` and `for` statements, and the special behavior of the `break` statement.

In the next unit we will look at functions, modules and packages, which are essential for well-structured code when programs become bigger and more complex.

Loops with the `while`-statement

While-loops are a straightforward way to implement conditional loops in Python. Here is a simple countdown example:

In [1]:

```
number = int(input("Please enter an integer number: "))
while number > 0:
    print(number)
    number = number - 1
```

```
Please enter an integer number:  10
```

```
10
9
8
7
6
5
4
3
2
1
```

While-loops have a simple basic form:

```
while <condition>:
    <do something>
```

That is, as long as (while) the condition is true, the body of the loop will be repeatedly executed. Just as in `if`-statements, the code forming the body of the loop is indented to mark that it is one block.

If the condition is never changed through the execution of a loop, this results in an infinite loop. This is a common mistake when programming. For example, if we forget to decrease the number in our countdown example:

In [ ]:

```
number = int(input("Please enter an integer number: "))
while number > 0:
    print(number)
```

Or if the decrement happens only after the loop body:

In [ ]:

```
number = int(input("Please enter an integer number: "))
while number > 0:
    print(number)
number = number - 1
```

Attention: In both these cases, the program will forever output the number the user entered. On the command line, pressing CTRL+C stops the execution of a program that is stuck in an infinite loop. In Jupyter notebooks, press the square Stop button (next to Run). In Spyder, both options work.

Sometimes it can be useful to work with infinite loops, though! For example, all software that is constantly running and ready to process user inputs (think for example of web servers) runs in a sort of infinite loop. We will see some cases later where it makes sense to use infinite loops.

Just as `if` -statements, also loops can be nested (contain other loops). For example:

In [4]:

```
number1 = int(input("Enter an integer number: "))
number2 = int(input("Enter another integer number: "))

while(number1 > 0):
    orig = number2
    while(number2 > 0):
        print(f"{number1} + {number2} = {number1+number2}")
        number2 -= 1
    number1 -= 1
    number2 = orig
print("Done.")
```

```
Enter an integer number:  2
Enter another integer number:  3

2 + 3 = 5
2 + 2 = 4
2 + 1 = 3
1 + 3 = 4
1 + 2 = 3
1 + 1 = 2
Done.
```

The following implementation of the number-guessing game is a slightly more complex example of a while-loop (as will be explained in more detail in one of the next lectures, the first two lines generate a random number between 1 and 10):

In [6]:

```
import random
number = random.randint(1,10)

number_guessed = False
print("Can you guess the number?")

while number_guessed == False:
    guess = int(input("Make a guess: "))
    if guess < number:
        print("You guessed too small!")
    elif guess > number:
        print("You guessed too big!")
    else:
        print("Yes!")
        number_guessed = True
```

```
Can you guess the number?

Make a guess:  5

You guessed too small!

Make a guess:  7

You guessed too small!

Make a guess:  9

You guessed too small!

Make a guess:  10

Yes!
```

[**Now is a good time to do Exercise 1.**]

The `break`-statement

The `break`-statement can be used to leave a loop before the loop condition becomes `False`. For example, we could extend the countdown example from above by a check every 5 steps if the user wants the countdown to go on or not:

In [8]:

```
number = int(input("Please enter an integer number: "))
while number > 0:
    if number%5==0:
        go = input("Do you want to continue? (y/n) ")
        if go == "n":
            print("Countdown stopped.")
            break
    print(number)
    number = number - 1
else:
    print("Ignition!")
```

```
Please enter an integer number:  20
Do you want to continue? (y/n)  y

20
19
18
17
16

Do you want to continue? (y/n)  y

15
14
13
12
11

Do you want to continue? (y/n)  2

10
9
8
7
6

Do you want to continue? (y/n)  n

Countdown stopped.
```

Or, as another example, we can use it to emulate "do-until" or "do-while" loops (which exist in some other programming languages) in Python, using a deliberately infinite loop (while `True`) and a conditional break in the loop body:

In [10]:

```
while True:
    number = int(input("You have to enter the right number to stop me: "))
    if number == 0:
        break
```

```
You have to enter the right number to stop me:  2
You have to enter the right number to stop me:  3
You have to enter the right number to stop me:  4
You have to enter the right number to stop me:  0
```

Loops with the `for`-statement

Loops with the `for` statement

We have seen while-loops as one way to implement loops in Python. While-loops evaluate arbitrary conditions to decide whether to enter to body of the loop (again) or not. For-loops are also in principle also conditional loops, but they are particularly made for iterating over collections of objects, in a "for each element in the collection do…" style. Here is a simple example:

In [11]:

```
for i in range(1,11):
    print(i)
```

```
1
2
3
4
5
6
7
8
9
10
```

The `range` function is used to obtain a so-called iterable range object comprising the numbers 1-10. The for-loop then goes through this iterable one by one, calling the next element `i` in each iteration and in the body of the loop simply printing the value of `i`. We will see more examples of iterable objects that can be used in for-loops next week. For now note that the basic form of a for-loop is:

```
for <element> in <iterable collection of elements>:
    <do something>
```

Note that for-loops can also be implemented as while-loops, but more “management code” is needed. The following code is equivalent to the example above:

In [12]:

```
i = 1
while i < 11:
    print(i)
    i = i + 1
```

```
1
2
3
4
5
6
7
8
9
10
```

Because of their design, for-loops do not (easily) lead to unintended infinite loops, so it’s preferable to use them when suitable objects to iterate over are present. Especially for nested loops this can greatly improve the readability of the code.

An interesting class of iterables is string, that is, we can easily use for-loops to iterate over all characters in a string. The following example shows how to iterate over all characters in a string and count the number of vowels in it:

In [2]:

```
text = "Just a small piece of text without a special meaning."

number_of_vowels = 0

for char in text:
    if char == "a" or char =="e" or char == "i" or char == "o" or char == "u" or \
        char == "A" or char == "E" or char == "I" or char == "O" or char == "U":
        number_of_vowels += 1

print(f"There are {number_of_vowels} vowels in \"{text}\".")
```

```
There are 18 vowels in "Just a small piece of text without a special m
eaning.".
```

[**Now is a good time to do Exercises 2 and 3.**]

Exercises

1. Reversed number-guessing with a `while` -loop

Write a program that reverses the number guessing game: The user enters a number between 0 and 100, and the program needs to guess it (without cheating!). Use a `while` -loop to let the program randomly guess a number until it finds the right one. Print out the number of attempts needed at the end. The output should look something like:

```
Please enter a number between 0 and 100 for the computer to guess: 85
The computer needed 290 attempts to guess the number.
```

Extra: Make the computer's guessing procedure smarter than purely random.

2. Text processing with a `for` -loop

Write a program that asks the user to enter a bit of text, and then counts the number of alphabetic and non-alphabetic characters in the text. Use a `for` -loop to iterate over the characters in the text. Print out the number of alphabetic and non-alphabetic characters at the end. The output should look something like:

```
Please enter some text:  Hello World!
Your input contains 10 alphabetic characters and 2 non-alphabetic character
s.
```

Hint: The statement `char.isalpha()` returns `True` if the character `char` alphabetic, and `False` otherwise.

3. Summing Up

Write a program that asks the user to enter an integer number n, computes the sum of all numbers from 1 to n, and prints the result. The output should look like:

```
Please enter an integer number: 5
The sum of all numbers from 1 to 5 is 15
```

Do you use a while-loop or a for-loop? Why?

4. Extra: Temperature Conversion Revisited

Extend the temperature conversion program from last week so that it asks the user to enter both a temperature value and the unit of the temperature (Celsius or Fahrenheit), and calculates the temperature in the respecive other unit. If an incorrect unit is entered, the program should give an error message and ask the user to try again. Furthermore, after having done a conversion, the program should ask the user if they want to convert another temperature value. The output should be something like:

```
Please enter temperature: 54
Is the temperature in Celsius (c) or Fahrenheit (f)? c
54.0 degrees Celsius is 129.2 degrees Fahrenheit.
Do you want to convert another temperature value? (y/n) y
Please enter temperature: 13
Is the temperature in Celsius (c) or Fahrenheit (f)? k
Unknown temperature unit, try again.
Please enter temperature: 13
Is the temperature in Celsius (c) or Fahrenheit (f)? c
13.0 degrees Celsius is 55.4 degrees Fahrenheit.
Do you want to convert another temperature value? (y/n) n
Okay, goodbye!
```

5. Extra: Text Analysis

Using what you have learned in the course so far, write a simple text analysis program that finds the (first) longest word in a text. It should work on any text, but you can use the “lorem ipsum” as an example:

```
# some random text
text = "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do \
eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad \
minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex
 \
ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate \
velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat
 \
cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id \
est laborum."
```

To check if a character c is an alphabetic character, you can call the isalpha() function on it: `c.isalpha()` . It will return True or False.

The output of your should report the longest word and its length, like this:

```
The longest word in the text is "reprehenderit" (13 characters).
```