Introduction to Programming W3

Iteration and expressions I.

2022/04/22

This class, we will learn about iteration in Python: for and while loops.

1 The "for" loop

One of the most useful *containers* in Python is called a **list**, a data structure that can group different data types together. Lists can be created using square brackets, and their elements can be accessed by their index, where indexing starts at 0 (same way we used indexing with strings). Their length is defined by the number of elements the list contain. We will learn more about lists in weeks 5 and 6. For now, try the following code:

```
my_list = ['Einstein', 'Curie', 'Newton', 1879, 1867, 1643]
print(my_list)
print(my_list[0]) #the first element in the list container
print(my_list[3])
print(type(my_list[1])) #checking the type of 'Curie'
print(len(my_list)) #how many elements does the list have?
```

The programming structure called **for loop** can be used to iterate over such containers as *iterables*- objects that can return one element at a time. The basic syntax of a for loop in Python is the following:

```
for <loop variable> in <iterable>:
    <loop body>
```

Note the colon at the end of the loop heading line, and that the loop body is indented. The statements in the body are executed once for each item in the iterable. The loop variable takes the value of the next element in the iterable at every iteration in the loop. Try the following code:

```
scientist_list = ['Darwin', 'Tesla', 'Galilei', 'Lovelace']
for scientist in scientist_list:
    print(scientist)
print("Name of scientists are printed!")
```

The loop/iteration variable is `scientist`, going over the iterable `scientist_list` one-by-one, and printing its values. For example, after printing the index 0 variable 'Darwin', the loop variable takes the value of the next element 'Tesla', and the loop body `print(scientist)` is executed on it. The process is continued until there is no more element in the iterable, then the next, *unindented* line `print("Name of scientists are printed!")` is executed, that is already outside of the loop body.
Try the following without deleting the assignment of `scientist_list` to see that strings are also iterables:

```
for character in scientist_list[1]:
    print(character)

for character in 'Programming':
    print(character)
```

Opposed to such *iterator-based for loops*, *range loops* can also be created in Python. The built-in `range()` function creates a *range object*, an iterable sequence of numerical values. Try the following code:

```
my_list = ['Darwin', 'Tesla', 'Galilei', 'Lovelace']

for i in range(4):
    print('iterating!')
    print(i) #notice how the value of i is changing
    print(my_list[i]) #notice that here, i is used to access the elements of my_list
print('Done.')
```

The name of the variable used for numerical iteration is not fixed, but using `i`, or `j` is conventional in Python. After printing 'iterating' at the beginning of every iteration, the current value of `i` is printed. Then the value of `i` is used as an index, to access and print the elements of `my_list` (`my_list[0]` at the first iteration, `my_list[1]` at the second, and so on). After there are no more values in the range object, the line outside of the for loop is executed. Notice that here as well, counting starts from 0. To print numbers 0, 1, 2, 3 (and using these as an index) the number provided is not 3, but 4. Try changing the number to 5, and check what kind of error you get!

The `range()` function actually takes three arguments: `range(start=0, stop, step=1)`, where the first argument is the first number in the generated numerical sequence, the second the last number in the sequence +1, and the last argument is the step size (integer difference between the numbers). The *default* value for `start` is 0, and the default for `step` is 1. The user do not have to provide default arguments, but can change them if needed. When there is no default value, the argument value must be provided (in this case, `stop`, such as in `range(5)`). The arguments have to be provided in the predefined order (e.g. `range(2,5,1)`, where the iteration starts from 2, ends at number 4, with step size of 1). Try the following code, and observe the output:

```
print("First for loop with range")
for i in range (4, 13, 2):
    print(i)
print("Second for loop with range")
for i in range(14,34,3):
    print(i)
```

2 The "while" loop

The **while loop** implements an indefinite iteration, because the number of times the loop body is executed is not specified explicitly but rather by a condition. The basic syntax of a while loop in Python is the following:

```
while <condition>:
    <loop body>
```

If the condition is `True`, then the loop body is executed. The condition is evaluated in every iteration (every time we go back to the while heading). The process is repeated until the condition becomes `False`, then the next unindented line is executed outside of the loop. Try the following code:

```
number = 10
while number > 4: #notice that the while loop heading also ends with a colon
    number -= 1 # decrements the variable by 1 at a time
    print(number)
    print(number > 4)
print('Done')
```

At line 3, the value of variable `number` is *decremented* by 1 in every iteration. Note that `number -= 1` is the same as `number = number - 1`, so variable `number` is reassigned. The same syntax can be used to *increment* a variable. For example, `number += 1` is the same as `number = number + 1`. Note that the order of operators is important; `number =+ 1` would just mean variable `number` is assigned to the number positive 1. Other operators like and * can be used the same way.

Remember that if the loop body does not modify anything related to the test condition, the code will result in an *infinite loop*.