Introduction to Programming W4

Iteration and expressions II.

2022/04/29

This class, we will learn about conditionals and continue learning about iteration in Python.

1 Conditional statements

Using the `if` statement, the code is only executed if a certain condition is evaluated as `True`. In case the condition is not satisfied, the code in the if block is skipped. The basic syntax is the following:

```
if <conditional statement>:
    <statement>
```

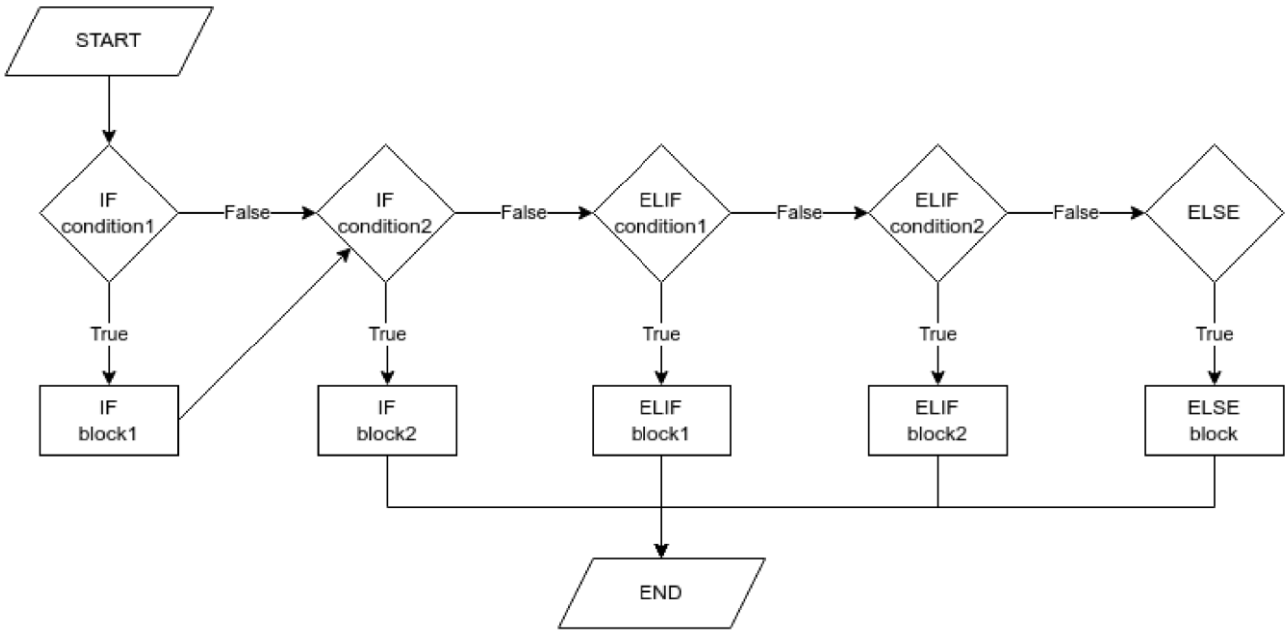
Note the colon at the if statement headline, and the indentation of the statement(s). For the conditional statements, comparison and logical operators (W2 material) can be used to obtain a Boolean value. Try the following code with different numbers:

```
number = 4
if number < 10:
    print(number, "is less than 10")

if number < 10 and number*2 > 7:
    print(number, "is less than 10 and multiplying it by 2 results in more than 7")
```

Additional note: Here, two different objects are printed within one print statement (not concatenation).
There are additional conditional clauses that can be used after an `if` statement: `elif` and `else`. `elif` ("else if") can be used to check for an arbitrary number of additional conditions if the previous `if` statement or `elif` statements are evaluated `False`. It is not rare to see multiple `elif` statements after the initial `if` clause. The `else` clause does not need a conditional statement; the `else` statement block runs if all conditions above it (one `if`, or one `if` and one or multiple `elif` statements) evaluated `False`. Try the following code and observe how the output changes based on the value assigned to `number`!

```
number = 10 #try the code with values 10, 25, 50, 150

if number > 100:
    print("number is greater than 100")
elif number == 50:
    print("number is 50")
elif number < 20:
    print("number is less than 20")
else:
    print("all above evaluated False")
```

The general flow of conditionals is the following (it is possible to include more than one `if` statements subsequently, although not recommended in most cases):



Conditionals can be *nested*, where indentation marks the level of nesting. Try the following code with different numbers (a positive, a negative, and 0), where nested conditionals are used to check if a number is positive, zero, or negative:

```
number = 17
if number >= 0: #checking if number is greater or equal than 0
    if number == 0: #if the number equals to zero
        print("zero")
    else: #every other case, connected to the if statement at the same indentation level
        print("positive")
else: #every other case, connected to the if statement at the same indentation level
    print("negative")
```

Additional note about conditionals using Booleans: Comparing a Boolean with `True` or `False` is redundant. Assume that raining is a bool type variable that is True if it is raining, and False otherwise:

```
if raining == True: #valid but redundant, since raining is already a bool
    print("Don't forget your umbrella!")
if raining: #this is enough
    print("Don't forget your umbrella!")
```

2 zip, enumerate

The built-in function `zip` is used to combine more than one iterables into one sequence to perform parallel iteration. For example, we can use it to iterate through two lists with two loop/iteration variables. Try the following code:

```
scientist_list = ['Einstein', 'Curie', 'Newton']
dob_list = [1879, 1867, 1643]
for scientist, dob in zip(scientist_list, dob_list):
    print("{} was born in {}".format(scientist, dob))
```

The built-in function `enumerate` can be used to create an iterator that has both the indices and values of a list. Try the following code:

```
scientist_list = ['Einstein', 'Curie', 'Newton']
for index, scientist in enumerate(scientist_list):
    print(index, scientist)
```

3 break, continue

`break` and `continue` statements can be useful to have more control over a for or while loop. `break` terminates a loop entirely, and `continue` skips the current iteration in the loop. It is important that `continue` skips the rest of the code

(inside a loop) for the current iteration only, while `break` *breaks out* of it entirely. Try the following code, observe the difference between `break` and `continue`, and note the different levels of indentation (very important).

```
scientist_list = ['Einstein', 'Curie', 'Newton', 'Darwin', 'Tesla', 'Galilei', 'Lovelace']
for scientist in scientist_list:
    if scientist == 'Tesla':
        break #from here, we go outside of the loop!
    print(scientist)
print ("for loop ended")

number_list = [1,4,6,33,78,99,3,150,12,1]
for number in number_list:
    if number >= 99: #we skip an iteration if the next number is greater or equal to 99!
        continue
    print(number)
print ("for loop ended")

n = 10
while n > 0:
     n += 2
     if n == 30:
         break #from here, we break out of the loop and the rest of the loop won't be executed!
     print(n)
print ("while loop ended")
```