SUMMER SCHOOL 2021

Introduction to Python Programming

Conditional Statements & Loops

Binjamin Barsch

Summary of commands used in this lecture

Command	Meaning
print()	**A Python function that prints text**
range()	**A Python function that creates a sequence of numbers that starts at 0 and increments by 1 and stops by the specified number.**

Conditional Statements & Loops

All programming languages support logical conditions and Python is no difffernt.

For example:

- Equals: `a == b`
- Not Equals: `a != b`
- Less than: `a < b`
- Less than or equal to: `a <= b`
- Greater than: `a > b`
- Greater than or equal to: `a >= b`

These conditions we typically use with conditional statements or **"`if`"** statements and loops. For example:

```
>>> current_year = 2021
>>>
>>> year = 2020
>>>
>>> if year < current_year:
... print("year is less than current_year")
  File "<stdin>", line 2
    print("year is less than current_year")
    ^
IndentationError: expected an indented block
>>>
>>> if year < current_year:
...  print("year is less than current_year")
...
year is less than current_year
>>>
```

I was planning to check if the year was smaller than "**`current_year`**". If it was, then I wanted to print "**`year is less than current_year`**". But I got an error! What is "**`IndentionError`**" all about. This is important to take note of. Most programming languages use some type of brackets to group a portion or block of code together that has various layers. Now Python uses spaces to group blocks of code. So, the "**`if`**" condition is layer 1, then layer 2 is the print function which has a space in the beginning which is dependent on layer 1. In the Python interpreter the three dots … notifies you that you are in a different layer of the block of code. Here are more examples:

```
>>> if 10 > 5:
...  print("yes this true")
...  print("this print function will work")
... print("this will give and error")
  File "<stdin>", line 4
    print("this will give and error")
    ^
SyntaxError: invalid syntax
>>>
```

So you see the third print statement gave an error as it did not have a space in the beginning. Now what happens if the condition isn't true. So if "**`10 > 20`**" then nothing would print as the condition would be false. So, we need to cater for two conditions. That is, the case where the "**`if`**" condition is true and false and we do that by using "**`else`**":

```
>>> year
2020
>>> current_year
2021
>>> if year > current_year:
...  print("true")
... else:
...  print("false")
...
false
>>>
```

Note that each print statement has a space as its in layer 2 and dependent on layer 1. And the "**`if`**" and "**`else`**" statement has no space as its in layer 1.

Now what if we want to check multiple conditions, that is having multiple if conditions. Then we use "**`elif`**":

```
>>> if year > current_year:
...  print("true")
... elif year == current_year:
...  print("the same")
... else:
...  print("false")
...
false
>>>
```

Let us print out the actual values so we can make sense of what we are trying to check instead trying to remember what year and **`current_year`** stands for:

```
>>> if year > current_year:
...  print("true: " + str(year) + " > " + str(current_year))
... elif year == current_year:
...  print("the same: " + str(year) + " = " + str(current_year))
... else:
...  print("false: " + str(year) + " < " + str(current_year))
...
false: 2020 < 2021
>>>
```

You can also combine conditional statements with logical operators like "**`and`**" and "**`or`**". And then we also get nested "**`if`**" statements too, or an "**`if`**" statement within another "**`if`**" statement and so on. But that we will cover at a later stage…on to loops.

Loops

This is the last section to learn that covers all the fundamental principles of Python coding. We will cover a "**`for`**" loop and a "**`while`**" loop. Take note that we still use the space to indent layers of the code block.

Let us say you want to print this year's date 3 times. You can type it out like this:

```
>>>
>>> current_year = 2021
>>> print(current_year)
>>> print(current_year)
>>> print(current_year)
```

This is fine, but very inefficient. The purpose of programming is to automate tasks and make things easier to do. So now let us use a loop to do the same thing:

```
>>>
>>> current_year = 2021
>>>
>>> for i in range(3):
...  print(current_year)
...  print(i)
...
2021
0
2021
1
2021
2
>>>
```

Let us first start with what "**`range(3)`**" does. This is a handy Python function that creates a sequence of numbers that starts at 0 and increments by 1 and stops by the specified number. So "**`range(3)`**" will give you: 0,1,2. It will not reach the number 3 but it will loop / or repeat the "**`print()`**" function 3 times.

The "**`i`**" variable is an arbitrary variable that we will use to do the looping. You can see that variable "**`i`**" increments as the loop progresses. Lastly the "**`for`**" command initiates the "**`for`**" loop.

Now let us try an example with trying to loop through a **list**:

```
>>> my_list = ["Summer", "School", 2021]
>>> for var in my_list:
...  print(var)
...
Summer
School
2021
>>>
```

So now we are using variable called "**`var`**" that will be assigned to each item in the **list** as it loops through the list. Pretty cool right.

One more example is a nested "**`for`**" loop, which is a "**`for`**" loop within a "**`for`**" loop:

```
>>> adj = ["red", "big", "tasty"]
>>> fruits = ["apple", "banana", "cherry"]
>>>
>>> for x in adj:
...   for y in fruits:
...     print(x, y)
...
red apple
red banana
red cherry
big apple
big banana
big cherry
tasty apple
tasty banana
tasty cherry
>>>
```

We see that the first layer has a "**`for`**" loop and the second layer has a "**`for`**" loop. And then within that "**`for`**" loop there is a third layer that uses the **print** function to show the values for "**`x`**" and "**`y`**". I hope this is confusing 😊

While Loop

Lastly, we will look at "**`while`**" loops. Be careful with "**`while`**" loops as you can get a never-ending while loop. Remember what happened to Neo…in the subway!

Ok it's not that bad…Let us do a similar example like the for loop above where we want to print the "**current_year**" three times:

```
>>> inc = 0
>>> current_year = 2021
>>>
>>> while inc < 3:
...  print(inc)
...  print(current_year)
...  inc = inc + 1
...
0
2021
1
2021
2
2021
>>>
```

You will notice that a "**while**" is like a "**for**" loop that is just split up into different sections. Another way of saying it is, a for loop is a more compact form of a while loop.

So what did we do above. First we have to declare some arbitrary variable that will do the condition check for the loop, that is if "**inc < 3**" then "**print(current_year)**". Since "**inc**" is initially 0 and is less than 3 then the condition is true and it will print the "**current_year**". The crucial part of this block of code is the "**inc = inc + 1**". That increments the value of "**inc**" to 1, and then 2, and then when it reaches 3 the "**while**" loop condition fails as "**inc**" is not less than 3 anymore and the "**while**" loop stops. If you do not write "**inc = inc + 1**" the "**while**" loop will go on forever, and ever…remember Neo…

Practice Exercises:

To help you understand some of these concepts here are some practice exercises:

1. Printing a pyramid with stars

We did a similar task in the first week of the summer school where we had to print a pyramid. Well in Python you can do that too!

We are going to make use of "**for**" loops. The first "**for**" loop in layer 1 of your code block will loop through the number of rows you want. Then in layer 2 of your code block we need another "**for**" loop to print the stars. Let us start with 5 rows:

```
>>> rows = 5
>>> for i in range(0, rows):
...     for j in range(0, i + 1):
...         print("* ", end="")
...     print("")
...
*
* *
* * *
* * * *
* * * * *
```

But this gives a triangle, we need to print spaces too. So, we will need another "**for**" loop in layer 2 to print spaces. For now, we will use circles instead of spaces to better visualize what we are printing:

```
>>> k = rows
>>> for i in range(0, rows):
...     for j in range(0, k):
...         print(end="o")
...     k = k - 1
...     for j in range(0, i + 1):
...         print("* ", end="")
...     print("")
...
ooooo*
oooo* *
ooo* * *
oo* * * *
o* * * * *
>>>
```

And now with spaces:

```
>>> rows = 5
>>> k = rows
>>> for i in range(0, rows):
...     for j in range(0, k):
...         print(end=" ")
...     k = k - 1
...     for j in range(0, i + 1):
...         print("* ", end="")
...     print("")
...
     *
    * *
   * * *
  * * * *
 * * * * *
```

Challenge: Can you print the pyramid upside down?

2. Even and Odd

Suppose you start with a certain number and you want to check if it is even or odd. If it is even you divide it by 2 and if it is odd you multiply by 2. This continues forever until the number is less than 1. How do we do that?

First off how do we check that a number is even or odd. Well if you divide a number by 2 and it gives a remainder of 0 then it is even, otherwise it is odd. To check for a remainder, we can use the modulo operator:

```
>>> 4 % 2
0
>>> 5 % 2
1
>>>
```

So, this shows 4 is even and 5 is odd as it has a remainder. Let us check this with a conditional statement:

```
>>> number = 4
>>> if number % 2 == 0:
...  print("This number is even: " + str(number))
... else:
...  print("This number is odd: " + str(number))
...
This number is even: 4
>>>
```

The last step is to put it in a loop and check when it goes less than 1:

```
>>> number = 4
>>>
>>> while number > 1:
...  print(number)
...  if number % 2 == 0:
...   print("This number is even: " + str(number))
...  else:
...   print("This number is odd: " + str(number))
...  number = number - 1
...
>>>
This number is even: 4
3
This number is odd: 3
2
This number is even: 2
>>>
```

Challenge: Can you get the same output by using a "`for`" loop?

Summary

This is the end of the lesson where we covered conditional statements and loops. On to the next lesson!
