Lecture 2: Pythonic logic and loops

Dr Benjamin J. Morgan[1] and Dr Andrew R. McCluskey[1,2]

[1] *Department of Chemistry, University of Bath, email: b.j.morgan@bath.ac.uk*
[2] *Diamond Light Source, email: andrew.mccluskey@diamond.ac.uk*

July 9, 2019

Aim

In this lecture, you will learn about logical operations, conditional statements, and for and while loops.

1 Logical operators

Previously, we observed how to use Python or a Jupyter Notebook as a simple calculator. However, programs become more useful when we are able to make the program more "intelligent" allowing it to perform different calculations under different circumstances. This ability relies heavily on the use of logical operators, as we shall see. A logical operator is code that returns a Boolean, either `True` or `False`. The logical operator `==` is one of the most common, and translates to *is equal to*, this operator will return `True` if the values on either side are the same, for example,

```
# The truth

print(14.0/2.0 == 7.0)
print(13.0/2.0 == 7.0)
```

This code will return `True` then `False`.

The equals operator is only one of many logical operators that is available in Python, some are given in Table 1. Each of these may be used in the same syntax as the equals operator.

Exercise

- Write code that will return the result of the following logical operations:
  1. $1 = 4$
  2. $10 < 15$
  3. $3.1415 \neq 3$

Table 1: Some logical operators available in Python.

Name	Mathematical Symbol	Operator
Equals	$=$	`==`
Less than	$<$	`<`
Less than or equal	$\leq$	`<=`
Greater than	$>$	`>`
Greater than or equal to	$\geq$	`>=`
Not equal	$\neq$	`!=`

2 Flow control

The `if` statement is one of the simplest, and most powerful, opeartions that Python can perform. This allows the code to apply different operations *if* certain criteria are `True`. An example of a Pythonic *if statement* is shown below,

```
# The if operator

if prior_meetings == 0:
    print("Hello World!")
```

Note that in Python the indentation is incredibly important (it is how the interpreter determines what is and is not part of the *if statement*. The above code asks the question, does the variable `prior_meetings` has the value `0`, and if it does print the string `Hello World!`

The `if` statement may be used in a more extended context, such as if the *logical argument* in the if statement is not True, an `else` can be included (or even an `elif`), as shown below,

```
# Five greetings to an old friend

if prior_meetings == 0:
    print("Hello World!")
elif prior_meetings == 1:
    print("Oh! You again")
elif prior_meetings == 2:
    print("Oh! You again")
elif prior_meetings == 3:
    print("Oh! You again")
elif prior_meetings == 4:
    print("Oh! You again")
elif prior_meetings == 5:
    print("Oh! You again")
elif prior_meetings > 5:
   print("Hello old friend!")
else:
   print("Meetings should be a positive number or zero")
```

3 AND and OR operators

In addition to the logical operators introduced above there are some others that it is important, and useful to be aware of. These are the `and` and `or` operators, which have important *but different* actions:

- The `and` operation returns `True` if both operations are true.
- The `or` operation returns `True` if either operations are true.

The code below gives and example of the syntax for these operations,

```python
# Using and and or

if 4 > 3 and 3 > 2:
   print("This is true")
elif 4 > 3 or 3 > 3:
   print("This is also true")
else:
   print("Nothing is true")
```

Exercise

- Use the `or` operator to reduce the `prior_meetings` code above to just 8 lines long. **Hint**: think about other operators from Table 1 to reduce the length of the code.

4 Loops

One of the best uses of programming (and computers) is to perform repetitive task over and over. For this we use *loops*, within Python there are two common types of loop:

- `for` loops iterate over a given sequence.
- `while` loops repeat as long as a certain logical operation is `True`.

An example of each of a `for` and `while` loop is shown below, both perform the same function,

```python
# For loop

for i in range(5):
   print(i)

i = 0
while i < 5:
   print(i)
   i = i + 1
```

Both of these code blocks will print the numbers 0 to 4, however the `for` loop is clearly more concise. Additionally, the `while` loop is more prone to accidently running an *infinite*. If you were to forget to manually iterate the variable `i` (this is the line `i = i + 1`), then the `while` condition would always be `True` and therefore the code would run forever within this loop. For this reason it is suggested that, where possible, you use a `for` loop over a `while` loop.

The `for` loop will iterate the variable (in the example above this variable is named `i`) through whatever sequence is given (this is `range(5)` above, which is equivalent to the *list* `[0, 1, 2, 3, 4]`). The sequence does not necessarily have to be a `range` command, it may be any `list` or `numpy.ndarray` (we will discuss these types later in the course). For example, in the code below we iterate though the first ten chemical element symbols,

```
# Printing the periodic table

elements = ["H", "He", "Li", "Be", "B", "C", "N", "O", "F", "Ne"]

for symbol in elements:
   print(symbol)

for i, symbol in enumerate(elements):
   print("The index of the list for {} is {}.".format(symbol, i)).
```

It is possible to use the `enumerate` command to count through the list during the loop, as shown in the second example above.

Exercise

- Recall from first and second year, that Python counts indices in a list from 0. How could the above code be adapted such that the correct atomic number will be printed?

4.1 Escaping loops

Sometimes it is computationally efficient to leave a `for` loop, to skip a particular value, under a certain condition. For this, the commands `break` and `continue` are available. The `break` command will exit the *inner-most* loop that is being carried out, while the `continue` command will skip the current value and jump immediately to the next. Examples of how these may be used are shown below, where the `len` function will return the *length* of the list,

```
# Finding the zero in a list

numbers = [1, 5, 7, 0, 2, 6, 2]
for i in range(len(numbers)):
   if numbers[i] == 0:
      break

print("The zero is at index {}.".format(i))
```

```
# Making all the negative values positive

numbers = [-2, 4, 1, -5, 2, 6, -3, -4]
for i in range(len(numbers)):
   if numbers[i] >= 0:
      continue
   else:
      numbers[i] = numbers[i] * -1
```

Note that the above examples are toy problems and there are more efficient way to carry-out these specific operations in Python.

5 Problems

The week there are two exercises to be carried out.

5.1 Interatomic distances

Write code that will take the x, y, and z coordinates of three atoms, and calculate the distances r between each pair, then print the distances along with the atoms that the distances are between. The equation for r between two atoms is,

$$r = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}. \tag{1}$$

This expression is widely used in atomistic simulations. **Remember**: Plan the procedure for your code before you start to write any Python. Use your code to calculate the distances between the atoms in the following molecules:

1. Atom 1: (0.1, 0.5, 3.2); Atom 2: (0.4, 0.5, 2.3); Atom 3: $(-0.3, 0.3, 1.7)$
2. Atom 1: $(-0.1, 0.5, 1.5)$; Atom 2: (0.2, 0.5, 2.6); Atom 3: (0.5, 0.5, 3.7)

Comment on the shape of each of the molecules.

5.2 Equilibrium constants

Write code that will calculate values of the equilibrium constant, K, for a given free-energy change over a range of temperatures. The program should ask the user for a free-energy value, ΔG or $Deltag$, and to specify the units for this (either kJ mol^{-1}, eV, or J). The initial temperature, T_{init}, final temperature, T_{final}, and temperature step size, T_{step} should also be entered by the user (in K). In order to learn more about how to do this with the `range` function, check the documentation online (`https://www.w3schools.com/python/ref_func_range.asp`). The equilibrium constant equation is,

$$K = \exp\left(\frac{-\Delta G}{RT}\right) = \exp\left(\frac{-\Delta g}{k_B T}\right) \tag{2}$$

where, $R = 8.314\,\text{J}\,\text{K}^{-1}\,\text{mol}^{-1}$, $k_B = 1.3806 \times 10^{-23}$ J, and $1\,\text{eV} = 96.485\,\text{kJ}\,\text{mol}^{-1}$.

When you check for what is typed, don't forget to check for upper-case as well as lower-case letters, as these characters have different ascii codes. You should also anticipate the possibility of the user entering a completely different letter (by mistake): what action would be appropriate in this event? Additionally, make sure that the user cannot make the temperature *unphysical* (e.g. less than or equal to zero). Again, remember to plan before you code.

Test the code using a temperature range from 100 K to 2000 K with a step size of 100 K, and with free energies of:

1. $-12.177\,\mathrm{kJ\,mol^{-1}}$
2. $-0.1452\,\mathrm{eV}$
3. $-2.6308 \times 10^{-20}\,\mathrm{J}$

Comment on the values at 300 K.