Lecture 1: Introduction to Python

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*

November 18, 2019

Aim

In this lecture, you will be introduced to Pythonic variable types, basic arithmetic, input and output (I/O) and intrinsic functions.

1 Introduction

The aim of this course is to develop skills in the user of computer programming (particularly in the Python programming language), building on the skills learned in the first and second year Computational Chemistry laboratory. You will then put these skills into practice, using Python to analyse chemical structures and perform quantum mechanical chemical calculations.

The Python programming language is one of the most popular programming languages in the world, ranking third on the TIOBE index (a measure of programming language popularity) in June 2019,[1] with the largest rate of change (it is becoming more popular over time). Additionally, it is probably the most popular programming language used in the chemical sciences. Recently, it was suggested that more than 7 % of all academic papers published in 2018 made mention of the Python (Figure 1).[2]

Python was first released in 1991, with one of the main design philosophies of the language being code readability. This readability is one of the driving factors to its adoption, along with some of the concepts introduced in this course, such as dynamical typing and powerful libraries like NumPy and Matplotlib. Since the early 1990s there have been three major versions of Python, with the most recent (and the focus of this course) being Python 3. Python 2 is still commonly found online and in libraries, however it is due to "retire" at the end of this year (check out `https://pythonclock.org` for a live countdown). Therefore, many packages are now dropping support for Python 2 and most practitioners would suggest new learners to start with Python 3.

1.1 Books

While this course is self-contained, the following books are particularly useful for those interested in learning more about Python:

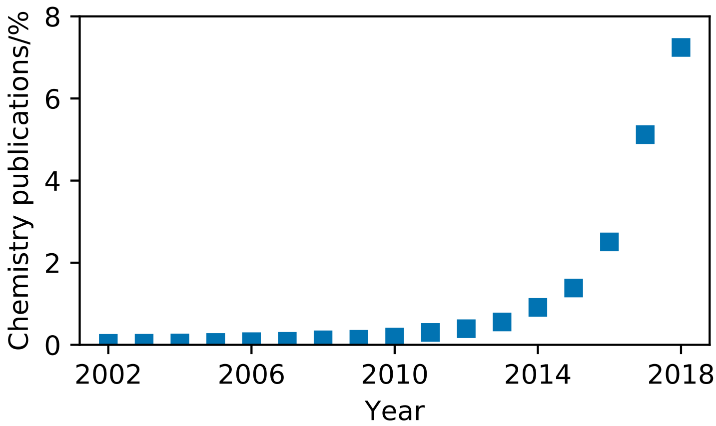

Figure 1: The percentage of "chemistry" publications that also mention "python", determined from the numbers of matching Google Scholar results.

- J. Vanderplas, *Python Data Science Handbook*, O'Reilly Media, Sebestopol, 2016. This book is available as a free e-book at: `https://jakevdp.github.io/PythonDataScienceHandbook/`.

2 Variable types

It is the case in many programming languages that *variables* can be assigned, a variable is a container used to store some data. It is possible to assign a variable as shown below,

```
# Variable assignment

banana = 1.
```

Not all variables are the same, and therefore variables may have different *types*, where different operations are possible depending on the type. Some examples of variable types that are present in Python include:

- **Integers** (`int`) – these are whole numbers (1, 2, 0, -3, etc.); there is no decimal point, and they can be positive, negative, or zero.
- **Floats** (`float`) – these are all *real* numbers (1.0, 3.14, 0.0, 6.28, etc.); any values that can be described using a decimal point.
- **Complex** (`complex`) – complex numbers should be familiar from mathematics, where they are typically written as $2 + 1i$, however in Python the i is replaced with a `j`, so $2 + 1i$ becomes `2+1j`.
- **String** (`str`) – a string is a textual variable such as a word or a sentence. These are written between single or double inverted commas, `'like this'` or `"this"`.

- **Boolean** (bool) – named for George Boole, who first defined an algebraic logic system in the 19th century, a Boolean is a *logical* variable type that may hold one of two values, either `True` or `False`.

The type of a given variable can be determined with the following command,

```
# Type determination

type(banana)
```

This function will return the type of the variable given as an *argument*.

Exercise

- Experiment with the different variable types and see if you can assign an example variable for each of the five outlined above.
- Determine the type of each of the following:
  1. `greeting = "Hello World!"`
  2. `pi = 3.1415`
  3. `life = 42`
- Consider the type difference between `1`, `1.`, and `1.0` as interperated by Python.

3 Variable assignment

It was mentioned above that a variable may be assiged with the following syntax,

```
# Variable declaration

banana = 1.
```

This establishes a name (`banana`) for some location in the computer memory, and places a value (`1.`) into that location. Once a variable has been assigned, it can be used in other parts of the code. For example consider,

```
# Reuse banana

apple = 5. + banana
```

Above, we have reused the banana variable assigned previously to create a new variable, `apple`.

Exercise

- Investigate the effect of changing the value of `banana`, after the variable `apple` has been defined.

4 Computer arithmetic

Python is able to basic mathematical operations natively (without the need for additional libraries to be loaded), some of these are shown in Table 2.

Table 1: The Python syntax for some basic mathematical operations.

Operation	Mathematical notation	Pythonic notation
Addition	$a + b$	`a + b`
Subtraction	$a - b$	`a - b`
Multiplication	$a \times b$	`a * b`
Division	$a \div b$	`a / b`
Exponent	a^b	`a ** b`

Using these basic tools alone, it is possible to use a Jupyter Notebook as a rudimentary calculator.

4.1 Order of operations

A single line of code can include many of the arithmetic operations outlined above, therefore it is necessary to establish a hierarchy, also known as the *order of operations*. Python follows the order of operations that should be familiar from mathematics, you may know this as BODMAS:

1. **B**rackets
2. **O**rder
3. **D**ivide
4. **M**ultiply
5. **A**ddition
6. **S**ubtraction

4.1.1 Exercise

- Without using the computer, and following the order of operations defined above, calculate the following:
    1. $24 \div (10 + 2)$
    2. $5 + 16 \div 2 \times 3$
    3. $(32 \div (6 + 2))^2$
- Now check your answers are correct using the computer.

4.2 Mixed mode operations

When the two operands are of the same type, the result of an arithmetic operation will also be of that type. However, the operation is on two operands of different types it is necessary to modify one of them before the operation is performed. For example, in the code below, an `int` is divided by a `float` and the type of the result will be a `float`,

```
# Divide an int by a float

mixed_type = 2 / 4.0

type(mixed_type)
```

This is because the `2` is converted to a `float`, so the operation becomes effectively `2.0 / 4.0`.

As of Python 3, if one `int` is divided by another `int` then both are converted to a `float`, meaning that, in the code below, the variable `both_int` will be a `float` with a value of `0.5`,

```
# Divide an int by a float

both_int = 2 / 4

type(both_int)
```

In Python 2, this would have returned an `int` with a value of `0`, however, with Python 3 the "floor division" operator (`//`) must be used to achieve this result,

```
# Divide an int by a float

floor_division = 2 // 4

type(floor_division)
```

5 Print and input (I/O) methods

Until now, you have been using the Jupyter Notebook to "print" the information. You may have noticed that the result of the final line in a given cell will be printed below it. However, if you would like to get information from a different part of a cell (or from within a larger program as we will see in later weeks), it is necessary to be able to print from the code. This is where the *print statement* comes in, in Python this looks like this,

```
# Print Hello World!

print("Hello World!")
```

This should output the string `"Hello World!"` when the cell is run (some of you may be aware that printing Hello World! is considered a right of passage in programming and is often the first thing someone will do when learning a new programming language).

Any of the types discussed above may be printed with the print statement, additionally it is also possible to use the print statement to insert numerical values into a string. We can even prescribe how the number is written, for example in the code below the information between the curly brackets tells the Python *interpreter* that the floating point number (`f`) should be written with two (`2`) numbers following the decimal point (`.`).

```
# More interesting printing

pi = 3.1415

print("pi to 4 decimal places is {} exactly!".format(pi))
print("pi to 2 decimal places is {:.2f} exactly!".format(pi))
```

More information about the Python `format` function can be found online (`https://www.programiz.com/python-programming/methods/string/format`).

In addition to printing, it is also possible to read information from the user. This is achieved using the `input` function, which will read the information given by the user as a *string* and is stored in the defined variable. The code below will test out the `input` and `print` functions,

```
# Who am I?

my_name = input("What is your name?")

print(my_name)
```

Since the information in `input()` is understood as a *string*, we may need to convert it; using `int()`, `float()`, or `complex()`.

6 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 2. Each of these may be used in the same syntax as the equals operator.

Table 2: 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	$\neq$	`!=`

Exercise

- Write code that will return the result of the following logical operations:

  1. $1 = 4$
  2. $10 < 15$
  3. $3.1415 \neq 3$

7 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")
```

8 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,

```
# 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 2 to reduce the length of the code.

9 Problems

9.1 Unit conversion

The final exercise of this week is to, in a single Jupyter Notebook cell, write some code that would allow a user to input a temperature in Fahrenheit and convert this to Celsius. For reference,

$$T(°\mathrm{C}) = \frac{5(T(°\mathrm{F}) - 32)}{9}. \quad (1)$$

You should consider how this may be broken down into *four* simple steps, write these steps out as an *algorithm* and then write your code from this algorithm. If you manage to complete this write another cell to convert from Fahrenheit to Kelvin.

9.2 Equilibrium constant

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 Δg, and to specify the units for this (either $\mathrm{kJ\,mol^{-1}}$, 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, $R = 8.314\,\mathrm{J\,K^{-1}\,mol^{-1}}$, $k_B = 1.3806 \times 10^{-23}\,\mathrm{J\,K^{-1}}$, and $1\,\mathrm{eV} = 96.485\,\mathrm{kJ\,mol^{-1}}$.

When you check for what is typed, remember that Python will understand upper case and lower case characters differently. 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.

References

[1] *TIOBE Index for June 2019*, 2019, `https://www.tiobe.com/tiobe-index/`, [Online; accessed 2019-06-25].

[2] A. R. McCluskey, *Introducing programming to undergradute chemists: and the tools we've developed to help them*, PyCON UK, 2018, `https://doi.org/10.6084/m9.figshare.7092167`.