

BCH2203 Python for Biochemistry - 1. Python Basics

Ramses van Zon

Winter 2022

Introduction

Topics

- Python basics
- Numerics and data representations
- Scientific Software Packages: NumPy and SciPy
- Genomics Analysis: BioPython
- File Input and Output
- Machine Learning
- Visualization

- One lectures per week, Wed 11:00 am - 12 noon
- Biweekly assignments posted on Wednesday
- Due the next Wednesday, then graded.
- Office hours Mon 11 am - 12 noon
- Completing this course means completing and submitting the 6 assignments.
- The average grade of the assignments is your course grade.
- Have to miss an assignment? Please inform us.

- We're going to get stuff done using a computer.
- But it will be stuff that is not be available in any existing application's menus.
- We will want to be able to repeat that same stuff again quickly.
- To accomplish that, we will need to put commands in some programming language in a file (script) and run it.
- We will use Python as the programming language.
- Apart from running scripts, Python also has an interactive command line.

- It is easy to learn.
- Particularly nice for data analysis.
- It's a “batteries included” language, i.e., a lot is possible with standard Python.
- In addition, there are many useful external packages for Python (Numpy, Scipy, Pandas, Matplotlib, Scikit-Learn, Biopython, ...)
- You can get a lot done with less effort compared to compiled languages like C++.

There are a few downsides, which we'll mention for completeness:

- Python code is slow.

If it's fast enough for your purposes, great, but some computations may require a compiled language.

Some python packages are written in a compiled language, so that can alleviate the slowness.

- Python code is dynamic.

This means some errors may only arise when running a code, there's no compiler helping you there.

- On your computer, open a “terminal”.
- This should give you some form of a **terminal prompt** (or “shell prompt”).
\$ will be used to denote the terminal prompt in these slides, regardless of the form of the terminal prompt for your system.
- At the terminal prompt, type “python”.
- This will give you a message regarding the version of Python.
(Try ‘python3’ instead of ‘python’ if the first number of the version is 2).
- It will also present you with a different looking prompt, the **Python prompt** “>>>”
- Following the Python prompt, type “print(9999+11111)”, and press enter.

```
$ python
Python 3.8.5 (default, Jan 11 2022, 19:51:10)
[GCC 6.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print(9999+11111)
21110
>>>
```

Didn't work? Here's what you will need for the course:

- [A Python installation](#)

Make sure you get Python 3 (not Python 2).

(e.g. from <https://www.anaconda.com>)

You could also work on SciNet's teach cluster:

```
$ ssh -Y USERNAME@teach.scinet.utoronto.ca
$ module load python/3
$ python
```

- [A plain text editor](#)

You need an editor that can save in **plain text format**.

E.g., nano, emacs, vi, vim, notepad, gedit, ...

- [A terminal or command prompt](#)

Because running a Python program is easiest from the command line ("shell").

E.g.: Bash, Mac Terminal, Anaconda Prompt, ...

- Graphical IDEs are not ideal for learning basic Python.
- The reason is that IDEs are big IDEs with lots of configuration options and lots of functionality.

When starting to program, that just distracts from the act of coding itself.

Furthermore, every IDE is different, changes and evolves, so for instructional purposes, it is better to stick with what works for everyone, everywhere, anytime.

- *But the command line is so much harder!*

It is different, so there are some things to get used to.

You may miss figuring out how to do something by looking at buttons and menus, pointing and clicking.

- 1) You can get documentation for nearly any function and package.

```
>>> help(print) # to get documentation about the print function
```

Anything after “#” on a line is ignored by Python, it’s a **comment** for your understanding.

*Someone told me/I read online that comments are unnecessary. **Don’t listen to them.***

- 2) If you have the name of some data structure, you can ask what type it is.

```
>>> type(__name__) # get the type, a.k.a. the class, of __name__  
<class 'str'>
```

- 3) Given some data structure, look inside it.

```
>>> dir(__name__) # look inside the __name__ structure
```

- 4) There’s Google (a search engine you may have heard of).

- A lot of the strength of Python is derived from the large body of available additional modules.
- These modules provide all kinds of functionality.
- There are many modules in the standard library that comes with Python (“batteries included”).

E.g. modules for GUIs, databases, random numbers, regular expressions, testing, . . .

- There are even more third-party modules available.
- The official repository for third-party modules is the Python Package Index (<http://pypi.python.org/pypi>) with over 100,000 packages.
- Most Python distributions come with the “pip” command, with which you can install packages from pypi.
(For Anaconda, you’d use the “conda”).

There are a number of ways to use Python:

1 Standard, non-interactive mode of Python

This requires an existing Python script.

Simply open a terminal and type

```
python <SCRIPTNAME>
```

and the code gets executed.

2 Standard, interactive mode of Python

Simply open a terminal and type `python`, and you get a prompt like `>>>`.

You can type commands at the prompt, they get executed, then you get another prompt.

3 IPython interactive mode

Requires IPython to be installed. Then type “`ipython`”, and you get a prompt that looks like “`In [1]:`”.

Features tab completion, command history, and special commands.

4 Jupyter notebooks

Input and output cells in your browser, with the Python back-end running possibly remotely. Harder to convert to scripts, and a bit more up-front work to get going.

<https://jupyter.scinet.utoronto.ca>

What does Python really do?

What happens when we type “`print(9999+11111)`” on the Python prompt?

- First, note that Python was waiting for input, and allows you to edit that input. It doesn't 'do' anything until you hit enter.

(in case of IPython, you can scroll through history and use tab completion, which are not 'doing nothing', but are still not doing Python)

- Once you hit enter, Python will check syntax, identifying functions, keywords, arguments, special characters, ...
- If it makes sense syntactically, it will then execute that command, i.e. translate it into (nested) function calls that at the lowest level are in machine code that the CPU understands.
- Python does this one line at a time, which puts it in the category of **interpreted languages**.

```
>>> print(9999+11111)
```

First action by Python: Syntax checking

- `print` is a name.
- It should be a function, because it is followed by parentheses.
- The argument of the function is `9999+11111`
- This is two 'literals' (numbers), separated by the plus sign, which is valid.

Second action by Python: Execution

- Store the integer 9999.
- Store the integer 11111.
- Call the `+` operator, with those integers as arguments.
- This “returns” a new integer.
- The `print` function is called with that new, temporary integer as an argument.
- Temporary integers are discarded.

Basic elements of the Python language

- You can give names to values in Python

```
>>> firstnumber = 9
```

We call this name-giving “assignment”.

- You can reuse a name:

```
>>> firstnumber = 9999
```

- The earlier value of `firstnumber` no longer has that name anymore.

Effectively, `firstnumber` has changed value.

- You can use variable instead of the value they refer too.

```
>>> print(9999)
9999
>>> print(firstnumber)
9999
```

- There are restrictions to the names: it can have letters, numbers, underscore, but cannot start with a number. No spaces, periods, brackets, etc., and they cannot be one of the reserved python keywords:

```
and assert in del else raise from if
continue not pass finally while yield
is as break return elif except def
global import for or print lambda
with class try exec
```

- Integer Number

```
>>> a = 13
>>> print(type(a))
<class 'int'>
```

- Floating Point Numbers

```
>>> b = 13.5
>>> print(type(b))
<class 'float'>
```

- Complex Numbers

```
>>> c = 1+5j
>>> print(type(c))
<class 'complex'>
```

- Strings

```
>>> d = 'Hello, world!'
>>> print(type(d))
<class 'str'>
```

- Bytes

```
>>> e = b'Hello, world!'
>>> print(type(e))
<class 'bytes'>
```

- Boolean

```
>>> f = (1==2)
>>> print(f)
False
>>> print(type(f))
<class 'bool'>
```

- With numbers, we can use arithmetic operators:

```
+ - * / % // **
```

E.g.

```
>>> 4 + 6//2
7
```

- We can use comparison operators:

```
== != < > <= >=
```

which result in a boolean, e.g.

```
>>> 4==1
False
>>> 3<5
True
```

- With booleans, we can do logic operators:

```
and or not
```

E.g.

```
>>> True and False
False
>>> (1==2) or (2==2)
True
```

- With strings, you can concatenate

```
>>> "Hello" + ", World"
"Hello, World"
```

and you can use comparisons:

```
== != < > <= >=
```

- Print is a function that prints its arguments to the terminal.
- Examples:

```
>>> print(100)
100
>>> print("Hello")
Hello
>>> a = 42
>>> print("Hello", "world!", a)
Hello world! 42
```

- You can print several things at once, by listing them in the parentheses to the print function, separated by commas.
- These 'several things' are called the arguments of the function.

The following code gives a “Good morning!” greeting if hour is less than 12.

```
>>> hour=11
>>> if hour < 12: print("Good morning!")
Good morning
>>> hour=16
>>> if hour < 12: print("Good morning!")
>>>
```

It does so for two case: hour=11 and hour=16.

- if followed by a **condition** and a colon will execute what is **after the colon** if the **condition** is met.

If we have **more than one statement** to do when the **condition** is met:

```
>>> hour=7
>>> if hour < 12:
...     print("Good morning!")
...     print("Would you like some coffee?")
...
Good morning
Would you like some coffee?
>>>
```

- *How did we enter a multiline piece of Python code at the >>> prompt?*
- Well, python noticed that the statement `if hour < 12:` requires more code, so it enters *multi-line mode*.
- Multi-line mode is indicated by `...`, and you can now continue typing code.
- To indicate that your multi-line input has been completed and should be interpreted and executed, you press Enter on an empty line.

If we have **more than one statement** to do when the **condition** is met:

```
>>> hour=7
>>> if hour < 12:
...     print("Good morning!")
...     print("Would you like some coffee?")
...
>>>
Good morning!
Would you like some coffee?
>>>
```

How does Python know which lines are part of the **statements that are executed** upon the **condition being true**?

- The if statement takes the next **code block**.
- The start of a code block is indicated by an **indented line**.
- That code block includes all lines that are indented in the same way.