BCH2203 Python for Biochemistry - 1. Python Basics

Ramses van Zon

Winter 2024

Introduction

About this quarter course

Topics

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

Format and Credit

- One in-person lecture per week, Wed 11:00 am - 12 noon
- Biweekly assignments posted on Wednesday
- Due the next Wednesday, then graded.
- Virtual office hours Monday 12 noon - 1 pm
- 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 let me know.

What is Programming?

- 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.

Why Python for Biochemistry?

- 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 addtion, 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.

Using the Python command line

- 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). We should be using python version or higher 3.9.
- 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
>>>
```

Required Software

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").

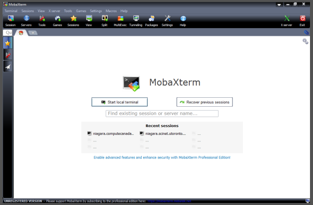
  E.g.: Bash, Mac Terminal, Anaconda Prompt, . . .

Short intro to the terminal a.k.a. console

How to get a terminal

On Windows

Get MobaXterm:


MobaXterm
Terminal Sessions View X server Tools Games Settings Macros Help
Session Servers Tools Games Sessions View Split MultiExec Tunneling Packages Settings Help X server Exit
Start local terminal
Recover previous sessions
Find existing session or server name...
Recent sessions
niagara.computecanada...
niagara.scinet.utoronto...
Enable advanced features and enhance security with MobaXterm Professional Edition!
UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here:

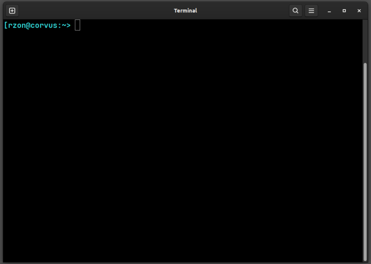
MobaXterm's local terminal runs the bash shell and comes with ssh and X11.

You can also use the Linux Subsystem for Windows.

Or use Anaconda Shell (no ssh).

On Linux

Find your terminal application.



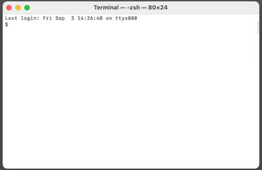
```
[rzon@corvus:~> 
```

The most common shell interpreter on Linux is bash.

It should have the ssh command.

On MacOS

Find your terminal application.



```
Last login: Fri Sep  3 14:26:40 on ttys000
$
```

The default shell is zsh or bash, depending on the MacOS version.

It should have the ssh command.

You need Xquartz for remote X graphics to work.

Command line interface

Command prompt

There is a prompt, *e.g.* `"[rzon@teach01:~]$"` after which you can type in commmands.

Any command you type at the prompt is read by a shell interpreter. Teach uses the bash shell.

Current directory

You are always "in" a current directory/folder in the file system tree. Your default directory, called your "home" directory, is where you start.

You can change to a directory with `cd DIRNAME`

`~` is a shorthand for that home directory.
`.` is a shorthand for the current directory
`..` is a shorthand for the parent directory.

Commands are either:

- built-in, or
- provided by executables in standard locations (encoded in the so called PATH variable), or
- executables of which the path is specified

Command examples:

- List the files in the current directory with `ls`.
- If the current directory contains an executable "first", execute it with the command `./first`.
- Connect to a different computer with `ssh`.

Command line arguments

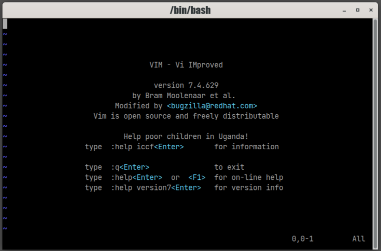
After a command, more words can be entered, the "arguments" of the command.

Tips on editing code

When logged into Teach with ssh, you cannot see the files on your computer.
Text-based editing of files in the terminal on Teach can be done using different applications.

vi

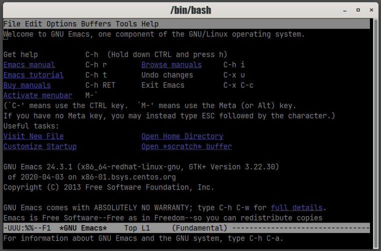
ubiquitous but not loved by all.



```
/bin/bash

~
~
~                    VIM - Vi IMproved
~
~                    version 7.4.629
~                by Bram Moolenaar et al.
~            Modified by <bugzilla@redhat.com>
~       Vim is open source and freely distributable
~
~               Help poor children in Uganda!
~      type  :help iccf<Enter>       for information
~
~      type  :q<Enter>               to exit
~      type  :help<Enter>  or  <F1>  for on-line help
~      type  :help version7<Enter>   for version info
~
~
~
~                                            0,0-1         All
```

emacs

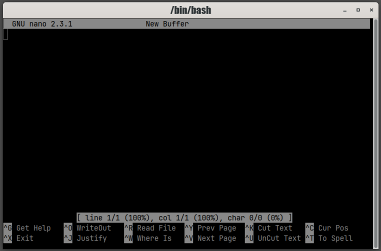
often available; not loved by all.



```
/bin/bash
File Edit Options Buffers Tools Help
Welcome to GNU Emacs, one component of the GNU/Linux operating system.

Get help           C-h  (Hold down CTRL and press h)
Emacs manual       C-h r        Browse manuals     C-h i
Emacs tutorial     C-h t        Undo changes       C-x u
Buy manuals        C-h RET      Exit Emacs         C-x C-c
Activate menubar   M-`
(`C-' means use the CTRL key.  `M-' means use the Meta (or Alt) key.
If you have no Meta key, you may instead type ESC followed by the character.)
Useful tasks:
Visit New File                  Open Home Directory
Customize Startup               Open *scratch* buffer

GNU Emacs 24.3.1 (x86_64-redhat-linux-gnu, GTK+ Version 3.22.30)
 of 2020-04-03 on x86-01.bsys.centos.org
Copyright (C) 2013 Free Software Foundation, Inc.

GNU Emacs comes with ABSOLUTELY NO WARRANTY; type C-h C-w for full details.
Emacs is Free Software--Free as in Freedom--so you can redistribute copies
-UUU:%%--F1  *GNU Emacs*    Top L1     (Fundamental) ------------------------------
For information about GNU Emacs and the GNU system, type C-h C-a.
```

nano

beginner friendly editor



```
/bin/bash
  GNU nano 2.3.1              New Buffer




                [ line 1/1 (100%), col 1/1 (100%), char 0/0 (0%) ]
^G Get Help   ^O WriteOut   ^R Read File  ^Y Prev Page  ^K Cut Text   ^C Cur Pos
^X Exit       ^J Justify    ^W Where Is   ^V Next Page  ^U UnCut Text ^T To Spell
```

Note: VS code and other GUI editors can be slow and tricky to setup on remote systems.

Why No Integrated Development Environment?

- 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.

How to figure things out in Python

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).

The Python Ecosystem

- 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`").

Different interfaces to Python

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?

Interpretation

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**.

Example: 9999+11111

```
>>> 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

Variables

- 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, and and they cannot be one of the reserved 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
```

  Also, avoid these:

```
int float str bool bytes complex set list dict tuple
range None False True abs bin pow round
```

Different types/classes of values in Python

- 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'>
```

What can we do with these data types?

- 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:

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

The `print()` function

- 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 `if` statement

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.

Multi-line input

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.

Code blocks

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.