Introduction to Programming (SCMP142)

Ramses van Zon

October 2023

SciNet
ADVANCED RESEARCH COMPUTING at the UNIVERSITY OF TORONTO

Section 1

Introduction

About this short course

The main point is to teach you the basics of programming!

We will be using the Python 3 programming language

Two one-hour sessions per week
Each sessions = short lecture + hands-on.

Topics

- Statements, expressions, variables, functions, objects
- Scripting
- Input and output
- Files and the file system
- Modularity

SciNet
ADVANCED RESEARCH COMPUTING at the UNIVERSITY OF TORONTO

Credit, certificates

- Completing this course counts for 8 credits towards a SciNet Certificate in Scientific Computing. (You would need 36 credits for the certificate.)
- Completing this course means:
  - Attending the sessions and taking the attendence test; and
  - Taking and passing the online test after the last session.
- Have to miss a session? Please inform us. In any case, you need to attend at least 5 sessions to get credit.
- This is *not* a course for UofT graduate credit.

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.

SciNet
ADVANCED RESEARCH COMPUTING at the UNIVERSITY OF TORONTO

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)

The command to use Python 3 may be `python3` instead of `python`.

You could also work on SciNet:

```
$ ssh USERNAME@niagara.scinet.utoronto.ca
$ module load NiaEnv/2022a python/3.11.5
$ python
```

- A text editor

  You need an editor that can save in **plain text format**.
  (e..g., nano, emacs, vi, notepad, gedit, vscode, . . .)

  Working on an Apple device? Make sure you switch off " Smart quotes " in the settings.

- A terminal or command prompt

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

  E.g.: Bash, Mac Terminal, A-Shell, Anaconda Prompt, . . .

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.

SciNet
ADVANCED RESEARCH COMPUTING at the UNIVERSITY OF TORONTO

Compute 9999 + 11111

Using a Graphical User Interface

- Start up your computer.
- On your computer, go to “Start”, “Applications”, “Calculator” (or your equivalent).
- Note: this should open a graphical calculator.
- On the keyboard, type `9999`.
- Then type `+`.
- Then type `11111`.
- Then type `=`.
- Read off the answer from the screen.
- Note: Alternatively, you can select the corresponding buttons on screen with mouse clicks.

Compute 9999 + 11111, again

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 if the first number of the version is 2 or if just `python` does not work).
- It will also present you with a different looking prompt, the **Python prompt**, either `>>>` or `In[1]:`
- After the Python prompt, type `print(9999+11111)`, and press enter.

```
$ python
Python 3.9.12 (main, Apr  5 2022, 06:56:58)
[GCC 7.5.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" f
or more information.
>>> print(9999+11111)
21110
>>>
```

Compute 9999 + 11111, and again

A first Python application

- On the same Python prompt, write

```
>>> f=open('app.py','w');f.write('print(9999+11111)');f.close()
```

- This creates a file called "app.py".
- That file contains just one line:

  `print(9999+11111)`

- Because it contains Python code, it is a Python application.
- To run the program, type `exit()` in Python, then type `python app.py`.
- Run the app again.

```
$ python
>>> print(9999+11111)
21110
>>> f=open('app.py','w')
>>> f.write('print(9999+11111)')
17
>>> f.close()
>>> exit()
$ python app.py
21110
```

```
$ python app.py
21110
```

Automation is what it's all about

- Automating the actions performed with a GUI is next to impossible.
- Once we had the text file "`app.py`", automation was easy.
- To create the file "`app.py`", requires some extra up-front work and knowledge (which we skipped over, and will usually do differently anyway)

Why No Integrated Development Environment?

- Although graphical, IDEs are not ideal for learning basic Python.
- The reason is that IDEs are big, 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 just 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.

SciNet
ADVANCED RESEARCH COMPUTING at the UNIVERSITY OF TORONTO

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. Or search the internet.

Hands-on #1: Installation Check

- Let's make sure we all have a working Python installation.
- We will be using Python 3.
- If you haven't installed Python yet, the easiest way to get it (currently) is probably anaconda.

- Open the terminal, type `python`, then, after the >>> prompt, type

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

- This should cause the number 21110 to be printed on the screen.

SciNet
ADVANCED RESEARCH COMPUTING at the UNIVERSITY OF TORONTO

The Python Ecosystem

- Although we will focus on the core Python language, the true strength of Python is 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 (https://pypi.org) 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` command instead).

Different interfaces to Python

There are a number of ways to use Python:

1. Standard, non-interactive mode of Python

   Open a terminal and type `python <SCRIPTNAME>`, and the code gets executed.

2. Standard, interactive mode of Python

   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 installation. Then type `ipython`, and you get a `In [1]:` prompt.

   Has tab completion, command history, 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.

   https://jupyter.scinet.utoronto.ca

   a. Jupyter Notebook
   b. Jupyter Lab
   c. VS Code jupyter emulation

Which should I use?

- Personally, I would recommend **IPython** for interactive work during this course.
- Just keep in mind some of IPython's special commands will not work in pure Python scripts.
- IPython has a special command to save and reload your session:

```
In[1]: a='Hello'
In[2]: b='World'
In[3]: print(a,b)
Hello World
In[4]: %hist -f mysession.py
In[5]: %load mysession.py
```

- The slides will nonetheless have the regular Python prompt >>>
  and everything will work both in regular Python and IPython.
  (In contrast to $ which stands for the terminal prompt.)

SciNet
ADVANCED RESEARCH COMPUTING at the UNIVERSITY OF TORONTO

Section 2

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 ("Parsing")

- `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.

Section 3

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. 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
```

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'>
```