Introduction to programming in Python

Christophe Pallier

4 September, 2017

Preliminaries

Why learning to program?

- "With computers, either you are their slave, our they are your slaves".
- It is hard to imagine a cognitive scientist who does not have a basic knowledge of computers and programming. Cognitive Science argueably started with the invention of computing by people like Alan Turing
- Because it is fun...

Coding

There are only a few really fundamental concepts in programming: data (and their types), variables, tests (control flow) and functions.

Once you know these concepts, learning a new programming language essentially entails:

- learning the syntax of the language
- learning the vocabulary, that is, apart from a few basic keywords, the functions made available by the language.

I recommend playing with the visual programming language *Scratch* (`http://scratch.mit.edu`) as it allows to acquire the fundamental concepts avoing frustrations with syntax and vocabulary (see `http://www.pallier.org/crash-course-in-programming-with-scratch.html#crash-course-in-programming-with-scratch`).

Python, unlike Scratch, is a *general purpose language*, in that it is suitable for (nearly) any kind of tasks. The drawback is that it takes longer to learn. Yet, contrary to most other programming languages, Python was not designed to be used exclusively by computer scientists. It strikes just the right balance between power and simplicity, and is well adapted to non professional programmers.

Your aim should not simply be to write programs that works !

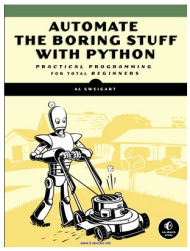
Your aim should be to write programs that work and *that you and other people can understand from reading the source code*

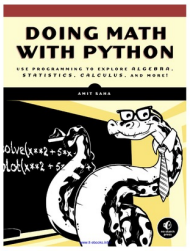
At a high level, coding is not unlike writing a text for humans: You must think about your reader and structure your code in a logical fashion, and use as appropiate names as possible for variables and functions. Also, you start from a first draft, and will modify it many times. A good design is when there is nothing more to remove.

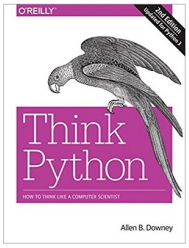
If this point is the only one you remember from this lecture, it would already be a success.

Ressources

- Books for beginners







- Al Sweigart *Automate the Boring Stuff With Python: Practical Programming for Total Beginners* available online at `https://automatetheboringstuff.com/`]
- Amit Saha *Doing Math with Python* (Chapter 1 available at `https://www.nostarch.com/download/Doing%20Math%20with%20Python_sample_Chapter1.pdf`)
- Gérard Swinnen *Apprendre à programmer avec Python 3* available online at `http://inforef.be/swi/python.htm`
- Allen B. Downey *Think Python. How to think like a Computer Scientist.* (2nd Edition, ver. 2.2.21) available online at `http://greenteapress.com/wp/think-python-2e/`
- On-line courses for beginners
- – Openclassroom's Python lecture `https://openclassrooms.com/courses/apprenez-a-programmer-en-python` (parts #1 and #2)

- Code Academy's Python course `https://www.codecademy.com/learn/learn-python`
- MIT's Gentle Introduction to Programming with Python `https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-189-a-gentle-introduction-to-programming-using-python-january-iap-2011/lectures/`
- More advanced Python

Once mastered the bases of python, you will be ready to embark into coding more ambitious and more useful programs. Here are some relevant ressources:

- for Numerical Computing, see the *Scipy lecture notes* `http://www.scipy-lectures.org/`
- for Data Analysis, Statistics, see Allen B. Downey's *Think Stats* `http://greenteapress.com/thinkstats2`
- for Science, see Allen B. Downey's *Think Complexity*. `http://greenteapress.com/wp/think-complexity/`
- General programming skills

To improve your general programming skills, for example, learn to use a version control system like git, see the lectures on the site 'Software Carpentry (Teaching basic lab skills for research computing)' `https://software-carpentry.org/`

Software

You should download and install:

- The Anaconda3 distribution from `https://www.continuum.io/downloads`
- The Atom text editor from `https://atom.io/`

Beware, there exist several versions of Python. We strongly recommend that you version 3 (version 3.6 at the time of this writing).

One can run Python interactively in the terminal with `ipython`, or within a browser with `jupyter notebooks`. One advantage of the jupyter notebooks is that they are *persistent*.

To write sligtly more complex programs (aka. scripts), or reusable modules, one needs to write Python code with a text editor, such as `atom` or `sublimeText`, or a development environment sych as `spyder` or `pyCharm`.

See "How to run Python"

We will use a slack workgroup (aip2017-info2.slack.com) to exchange during the lecture. Request an invitation if you have not received one.

Self-study program

Level 0

- Concepts
- Python Basics (numbers, strings, expressions, variables, flow Control: `if`, and `while` loops).
- Ressources
- Chapter 1 of Amit Saha *Doing Math with Python* (available at `https://www.nostarch.com/download/Doing%20Math%20with%20Python_sample_Chapter1.pdf`)
- Chapters 1 and 2 of *Automate the Boring Stuff* ou Chapitres 2-5 d'*Apprendre à Programmer avec Python*
- Exercices

1. Write code that asks for your name and greets you, e.g. if your name is 'Chris', it should write 'Hello Chris' (use Python's functions `input` and `print`); then modify it so that if you type 'Satanas', it would print 'Vade retro Satanas!'
2. write code that asks the user to type some text, and when enter is pressed, prints the number of characters (hint: use the funtion `len`)
3. write code that asks the user for a price and print the price with 33% reduction. Try to find a way to print the new price with only to 2 decimal places.
4. write code that computes your income tax, obeying the following rules:
   - All taxpayers are charged a flat tax rate of 20%.
   - All taxpayers are allowed a $10,000 standard deduction.
   - For each dependent, a taxpayer is allowed an additional $2000 deduction.
5. Write code that repeatedly asks for number and prints whether there are odd or even (the modulo operator is `%`). The loop should stop when the user enters '0'.
6. Write code that computes the sum of the first 1000 integers (tip: use a `while` loop).

7. The following program draws a triangle. Modify it to draw a square, then any regular polygon with 'n' sides.

```
import turtle
tt = turtle.Turtle()
tt.pendown()
tt.forward(100)
tt.left(120)
tt.forward(100)
tt.left(120)
tt.forward(100)
```

8. Write code that computes the sum of the squares of the first 1000 integers (tip: use a `while` loop).

9. write code that prints the first elements of the Fibonnaci series (u(0)=0; u(1)=1; u(n+2)=u(n+1)+u(n)) which are lower than 1.000.000.000

10. Write a program that estimates the number PI/4 by drawing pairs of random numbers between 0 and 1 (`x, y = random.random(), random.random()`)and computes how many of them represent points within the unit circle (`x**2 + y**2 <= 1`).

11. Given a binary number represented as a string of 0s ans 1s (e.g. '101'), compute its value and print it.

12. Given an integer number, convert it into its binary representation, as a string of 0s and 1s.

Level 1

- Skills
- Running Python in a terminal with ipython;
- Using a text editor to write scripts.
- Concepts
- Modules, Functions, scope of variables (Chapter 3 of *Automate the Boring Stuff*). Objects and Methods.
- Exercices

1. write a function that, given `r`, returns the volume of a sphere of radius 'r' (which is equal to 4/3 pi r^3).

2. Write a *script* that asks for your name and then greets you.

3. write a function that computes the factorial of an integer number (`factorial(n) = n(n-1)(n-2)...x 2 x 1`)

4. "Un magasin propose des CD à 15€ et des des DVD à 20€. Pour tout achat d'au moins 4 CD, il est consenti une réduction de 20% sur les CD. Pour l'achat d'au moins 3DVD, il est consenti une réduction de 15% sur le DVD. Construire un programme permettant d'obtenir le prix à payer suivant le nombre n de CD et m de DVD achetés."

5. Write a program to compute the greatest common divisor (GCD) of two numbers.

6. Write a program that plays the "guess a number" game: the computer selects a random number between 1 and 100, asks you to make a guess and tells you if the guess is correct, too low or too high.

7. Write a program that plays "guess a number" where *you* think about a number between 1 and 100 that the computer must guess.

Level 2

- Concepts
  - Lists, tuples and Dictionaries,
  - Mutable and inmutable objects
  - Iterations, String manipulation.

Read Chapter 4 (Lists), 5 (Dictionaries), 6 (Strings) of *Automate the Boring Stuff*

- Exercices

1. Write code, that asks the user for a text, and then prints in reverse (e.g. inputting 'Hello' should produce 'olleH').

2. Two taxi companies propose differents pricing schemes: "Tarif A: prise en charge de 4.80€ puis 1.15€ par km parcouru. Tarif B: prise en charge de 3.20€ puis 1.20€ par km parcouru." Ecrire un programme permettant de décider quelle compagnie choisir en fonction du nombre de kms à parcourir.

3. Given the function:

```
python    def is_factor(a, b):        return b % a == 0
```

Write a program that lists the prime numbers between 1 and 1000.

1. Write a program that determines prime numbers between 1 and 1000 using the Sieve of Eratoshenes `https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes`
2. Write a program that evaluates arithmetic expressions written in Reversed Polish Notation (`https://en.wikipedia.org/wiki/Reverse_Polish_notation`). For example the string '3 4 5 * +' would evaluate to 23.
3. Write a program which draws Koch's snowflake (`https://en.wikipedia.org/wiki/Koch_snowflake`). You may use, for example, the `turtle` module.

Level 3

- Concepts
- File Input/Output (Chapter 8 *Automate the Boring Stuff*)
- Exercices

1. Write a function that reads a text file and count how many words (tokens) it contains (we consider that a word is simply anything between two whitespaces).
2. Write a function that reads a text file and count how many different words (types) it contains.
3. Write a function that computes the number of occurrences of each word type in a text. First, consider that tokens are anything separated by whitespaces. In a second step, improve you function to clean the punctuations, ignore case changes, etc...

Projects

1. Write a simulator of a *Rodrego Register Machine* (`https://sites.tufts.edu/rodrego/`). The input is a list of instructions (no need for line numbers).
2. program one or several of the Sudoku solvers (`https://en.wikipedia.org/wiki/Sudoku_solving_algorithms`)
3. Program a cellular Automata (See Chapter 6 of *Think Complexity*. `http://greenteapress.com/wp/think-complexity/`)
4. program a Pong game (`https://en.wikipedia.org/wiki/Pong`) or a breakout game `https://en.wikipedia.org/wiki/Breakout_(video_game)` . You may use `pygame`, `pySDL2` or `pyglet`.

5. Write a program implementing a psychological experiment — the Simon task (`https://www.researchgate.net/figure/266736582_fig6_Figure-2-Trial-structure-and-stimuli-A-four-alternative-Simon-task-in-which-two-colors`). The exeriment is a succession of trials where either a green or a red disk is displayed. The user must press a left button for green disk, and a right button for red disk. Their reaction time is measures, in milliseconds. The trick is that the disks appear either at the left or at the right of a central fixation point. Althought location is irrelevant to perform the task, one typically observed an increase in reaction times when the side of display and the side of response are incongruent. To program this experiment, you should use the `expyriment` module.

6. Write a program implementing a psychological experiment: The *Attentional Network Task* (ANT), using the `expyriment` module. (see `https://www.researchgate.net/figure/265392381_fig1_Schematic-illustration-of-the-attention-network-test-ANT-as-applied-in-our-study-A`)