Python for Open Neuroscience

Lecture 1: Computers and programs

Luigi Petrucco

2025-02-20

Outlook

- A brief recap
- What do computers ultimately do?
- How to talk with a processor
- Make our life easy with programming languages

Von Neumann architecture

- The layman version

A program is an output recipe!

- ingredients: input data variables
- instructions: program lines

A recipe for tomato soup:

- ingredients (input data):
  - 1 onion
  - 1 kg tomatoes
  - oil

- instructions (program):
  - chop tomatoes, onions
  - fry onions in a pot
  - cook for 10 minutes
  - add tomatoes in the pot, adjust salt and pepper
  - cook for 20 minutes

Programming humans and computers

- lots of ambiguity, both in terms of actions and references!

How talk to computers actually look like

- ingredients (input data):
  - `a = oil`
  - `b = onion`
  - `c = tomatoes`
  - `d = salt`
  - `e = pepper`

- instructions (program):
  - 1. `chop b`
  - 2. `chop c`
  - 3. `fry b using a`
  - 4. `add c to the pot`
  - 5. `add d and e to the pot`
  - 6. `taste soup`
  - 7. `if taste is bad -> go back to step 5`
  - 8. `cook for 20 minutes`

Defining variables

Variables are “aliases” that we use to data in our program. Variables are basically earmarked locations in the computer memory where data is stored!

Variables can be:

- just a name for a value we enter manually (e.g. `a = 1`)
- the result of an operation or a function (e.g. `b = a + 2`, or `c = log(a)`)

Variable types

- In a program, variables have types:
    - boolean values
    - alphabetical characters (important to distinguish program text from data text!)
    - numbers

How many bits do we need *in principle* for each type?

Simple operations

- variables can be manipulated with simple operations
- mathematical operations (e.g. `+`, `-`, `*`, `/`)
- boolean operations (e.g. `==`, `!=`, `>`, `>=`, `<`, `<=`)
- . . . other more complicated operations for specific types

Operations results

- operations take existing variables (inputs) and produce new variables (outputs)
- the output can be assigned to a variable, or not

```
a = 1
b = 2
c = a + b  # what is the type of c?

d = a > b  # what is the type of d?
```

Program steps

- in a program we will always find a sequence of “atomic” instructions
- These instructions are separated by language-specific “delimiters” (e.g. new line, semicolon, etc.)
- Blocks of instructions (at the same level) are executed one after the other

Flow controls

- A program (algorithm) can be represented as a flow chart. . .
- . . . but we can only write a program one line after the other!
- We need to control the flow of the program, i.e. the order in which the instructions are executed

Conditional statements

Sometimes we want our program to act differently based on some conditions! This is where we use *conditional statements*!

Example in recipe

```
- ...
- 5. `add d and e to the pot`
- 6. `taste soup`
- 7. `if taste is bad -> go back to step 5`
- ...
```

Example in Python

If we want to compute an absolute value, for example:

```
number = -15

if number < 0:
    absolute_value = -number
else:
    absolute_value = number
```

Loops

Another frequent control structure is the *loop*, to repeat a line of code multiple times

If we had to detail more how chopping works in the recipe, we could write:

```
- 0. take onion
- 1. raise knife
- 2. cut slice
- 3. go back to 1
```

Break free from the loop

With for loops it is easy to get stuck in an infinite loop! We have to give more specifications:

The “do n times” (*for*) loop:

```
- 0. take onion
- 1. for 20 times:
    - 1.0 raise knife
    - 1.1 cut slice
    - 1.2 go back to 1.0
```

The “do until” (*while*) loop:

```
- 0. take onion
- 1. until onion is not all cut:
    - 1.0 raise knife
    - 1.1 cut slice
    - 1.2 go back to 1.0
```

A python example: `for` loop

```
i = 0
to_add = 5

for n in range(3):  # this is the syntax do do stuff n=8 times
    i = i + to_add

print(i)
# what is the final value of i?
```

A python example: `while` loop

```python
i = 0
to_add = 5

while i < 16:  # check if i < 16 and do stuff if true
    i = i + to_add

print(i)
# what is the final value of i?
```

Make your life easier with functions

- functions are a way to group instructions together
- they can take input variables and return output variables
- they can be used as atomic units in your program

A function example

Let's go back to our chopping instructions. We have to do it for both onions and tomatoes.

Without functions:

```
- 0. take onion
- 1. until onion is not cut:
    - 1.0 raise knife
    - 1.1 cut slice
    - 1.2 go back to 1.0
- 2. take tomatoes
- 3. while tomatoes are not cut:
    - 3.0 raise knife
    - 3.1 cut slice
    - 3.2 go back to 3.0
```

With functions:

```
- 0. define function `chop(x)`:
    - 0.0 take x
    - 0.1 until x is not cut:
        - 0.1.0 raise knife
        - 0.1.1 cut slice of x
        - 0.1.2 go back to 0.1.0
- 1. chop(onion)
- 2. chop(tomatoes)
```

In Python

- There are a bunch of built-in functions in Python! For example:

```
a = abs(-15)  # compute absolute value
```

Custom functions

- We can also define our own functions!

```
def my_absolute_value(x):
    if x < 0:
        absolute_value = -x
    else:
        absolute_value = x
    return absolute_value  # this is how we send values out of the function
```

Mental code chunking

- the most important and challenging skill to acquire:
- split in your mind the atomic operations that are executed in a program
- understand the flow of the program

Questions?

That's all!