Introduction to Programming Using Python

Applications in Computational Biology

Anastasis Oulas

Evangelos Pafilis

Jacques Lagnel

Why programming - Solving a problem

- A problem can be defined as a situation that requires a solution, however its solution is not known nor obvious.
- In order to deal with a specific problem one must first comprehend it. This is a function of two aspects:
  - The correct wording by its creator
  - The correct interpretation by the one who will solve it.

Example of a problem

- Let's say you have discovered a very interesting segment of DNA
- Perform a search of Genbank and other data sources using BLAST.
- Although you find a few related sequences the public genetic databases are growing daily and rapidly.
- You would like to perform your searches every day.
- But this could take an hour or two each day! Luckily, you know Python.
  - write a program that automatically conducts a daily BLAST search of Genbank for your DNA sequence, compares the results with the previous day's results, and sends you email if there has been any change.
- This program is so useful that you start running it for other sequences as well, and your colleagues also start using it.
- Within a few months, your day's worth of work has saved many weeks of work for your community.

The stages of solving a problem

Comprehension

↓

Analysis

↓

Solution

Problem solving using Computers

- Computers act synergistically to complement human activities they don’t have the capacity of independent thinking. People utilize them for problem solving due to their ability to:
  - Perform complex calculations
  - Repeat multiple processes
  - Execute calculations with high speed
  - Process large amounts of data

Algorithms - Definitions

- Definition
  - An algorithm can be define as an effective method for solving a problem expressed as a finite sequence of steps.
- An algorithm should comply with the following criteria:
  - Must have an **input** in the form of some type of data.
  - It should be **effective** i.e. contain simple execution commands.
  - Each command should be executed with **definiteness**, i.e. division by 0 not a definite command.
  - Should terminate after a **finite** sequence of steps
  - Produce an **output** (results).

A Simple Algorithm

- Eg. A simple set of steps:
- Running a BLAST > query sequence against GenBank > get results

- Eg A simple set of steps that satisfy certain conditions along with alternatives
- I want to run a blast on sequence of interest. Is there any difference to previous results > Yes > Send email with updated results, No> don’t send email

- Eg I want to run blast on a daily basis
- Every morning
    - query sequence against GenBank > get results
    - Is there any difference to previous day results?
        - yes: send email
        - No: Don’t send email and try again tomorrow

Course Introduction and Objectives

- Python
  - Powerful, flexible and easy to use
    - No semi-colons, brackets or other “strange” characters
  - Good candidate for building software tools and applications for life sciences
  - Good candidate for researchers, support staff, software developers, etc…
  - A lot of examples from computational biology in Python

- Course: Intended for people with no programming experience at all

Python

- In the tutorial:

  Demo with instructions for installing and running Python on your machine.

Installing Python

- Python can be downloaded from:

  http://python.org/download/

  (select version 3.2)

- Installers are available for OS X, Linux Windows. (In Linux and recent OSX Python is included by default)

Running Python in Windows

- We will make use of a freely available text editor **“notepad++”** which supports Python to write source code http://notepad-plus-plus.org/
- Python files have to be saved with the suffix **.py** (eg Ex1.py)
- To execute the source code in the file we will
  - open a ***command*** window (Windows Start button> Run> **cmd**)
  - change to the directory containing the source code and simply type: *python filename.py.*

- Note – if your computer complains about python, ask a tutor to assist you)

Running Python

- The term *command line* refers to where you type commands to a “shell”—in particular, a Unix shell such as tcsh or bash or a Windows command window

Running Python

- Python can also be run *interactively*, it prints some information about its version. Then it repeats a cycle in which it:
  - Prints the *prompt* >>> to indicate that it is waiting for you to type something
- *However we will* ***not*** *be using this mode!!*

Primitives

- Variables
- Operators
- Expressions

Variable Names

- Accepted variable names:
  - Can be any Latin character (e.g. x)
  - Can not contain numeric operators (-,+)
  - Can not be the same as a reserved python names i.e. **print**
  - Can not start with a numeric value.
  - Names are **case-sensitive!**

Variable types

- Three types are used far more frequently than others:
  - **Numerical**
    - *integer*
    - *float*
  - **Character-based**
    - *String*
  - ***Logical*** **(Boolean)**
    - *True/False*

Variable types - Integers

- There’s not much to say about Python integers. Their type is **int**, and they can have as many digits as you want. They may be preceded by a plus or minus sign. Separators such as commas or periods are not used:
- Examples:
  - 14
  - −1
  - 1112223334445556667778889990000000000000

Variable types - Floats

- “Float” is an abbreviated version of the term “floating point,” which refers to a number that is represented in computer hardware in the equivalent of scientific notation. Such numbers consist of two parts: digits and an exponent:
- Examples:
  - 2.5
  - 2e4

Variable types - Strings

- Strings are series of characters. Their type is **str**. x
- A string is enclosed in a pair of **single** or **double quotes**.
- DNA,RNA or amino acid sequences can be represented as strings: e.g. 'MNKMDLVADVAEKTDLSKAKATEVIDAV'

Variable types - Booleans

- There are only two Boolean values: **True** and **False**. Their type is **bool**.
- NB: Python names are “case-sensitive,” so true is not the same as True.

Operators

An *operator* is a symbol that indicates a calculation

Numeric Operators

- Some numeric operators are
  - **+ , - , * , /** , ** (power)
- There are three operators for the division of one integer by another:
  - / produces a float,
  - // (*floor division*) an integer with the remainder ignored, and
  - % (*modulo*) the remainder of the floor division.
- **Examples:**
  - **11 / 4 =** 2.75
  - **11 // 4** *# "floor" division*
    2
  - **11 % 4** *# remainder of 11 // 3*
    3

String Operations

- A new string can be produced by ***concatenating*** two or more existing strings. The result is a string consisting of all the characters of the first operand followed by all the characters of the second.
- A one-character substring can be extracted with ***subscription*** and a longer substring by ***slicing*** (not used in today's tutorial)

String Operations - Operators

- There are four operators that act specifically on strings:
- **+** : concatenation of two strings
- **in**, **not in** (containment or not) (will be discussed later in the course)
- ***** : repeats a string a given number of times

Operators - Comparison Operations

Six *comparison operators* that return Boolean values:

==	equal
!=	not-equal
<	less than
<=	less than or equal
>	greater than
>=	greater than or equal

Operators - Logical Operations

- The classic Boolean operators are
- **not**, **and**, and **or**.

- Can be combined with the comparison operators
  - Eg 3 < x < 8 can be represented

```
      x < 8 and x >3
```

Expressions

- We have seen that an *operator* is a symbol that indicates a calculation using one or more *operands*
- An operand can be any variable or constant (eg the number 2)
- The combination of the operator and its operand(s) is an *expression*.
- Examples:
  - 2+2   (numerical expression)
  - x < 8 (logical expression)
  - 'AC' + 'TG'   (String concatenation)

Assigning values to variables

- The result of an expression can be assigned to a variable using the **=** sign.
- Examples: y = 1, x = 2+y

Hierarchy of Numerical operations

- In decreasing priority: (), //, %, **, *, /, +, -
- Parenthesis can be used to group numerical operations
- Eg $6x^4 - (9 - y)^3$ can be written as:

```
6*x**4  - (9 - y)**3
```

Comments

- When a # symbol appears on a line of code, Python ignores it and the rest of the line.
- Text following the # is called a *comment*.

Basic I/O functions

- prompt user for input from the keyboard

```
x = input('Enter a number: ')
```

- Print a variable’s value to the screen

```
print (x)    #NB: don’t forget the parenthesis
```

Basic I/O functions

- Example script file printing the name of the course

```
name = 'programming course'
print (name)
```

Practical

- https://sites.google.com/site/pythoncompbio/
- 01_SequentialProgramming_Exercises.doc