Python-ohjelmointia Biotieteilijöille Avoimessa 2019

Janne Ravantti & Heidi Hulkko

janne.ravantti@helsinki.fi

Luennot suomeksi ja materiaali englanniksi

=>

"Python's documentation, tutorials, and guides are constantly evolving." (https://www.python.org/doc/)

https://jakevdp.github.io/PythonDataScienceHandbook/

https://en.wikibooks.org/wiki/Non-Programmer%27s_Tutorial_for_Python_3

https://stackoverflow.com/questions/40557910/plt-plot-meaning-of-0-and-1

Slides & example data in Moodle

Schedule & places

https://courses.helsinki.fi/fi/ay930101/128314088

Ask questions!

https://insights.stackoverflow.com/survey/2019?utm_source=so-owned&utm_medium=announcement-banner&utm_campaign=dev-survey-2019

& use search engines + stackoverflow, but be careful out there!

Goals for the course:

1) Tools to do bioinformatics with Python on your own!

2) Pointers to some useful libraries

3) Practise Python programming

Outline

1. Python-programming warm up
2. Anaconda Distribution
3. Python as an Integration language
4. Libraries...
   - 4.1. Numpy & Scipy
   - 4.2. Matplotlib
   - 4.3. Pandas
   - 4.4. Biopython
   - 4.5. ...
5. Recap

Programming requires peculiar way of thinking
(but it can be learned!)

Good* way to learn programming is to program!

*The Best?

Bioinformatics & Python?

---------- Python ----------

Comp.sci
Statistics
Mathematics

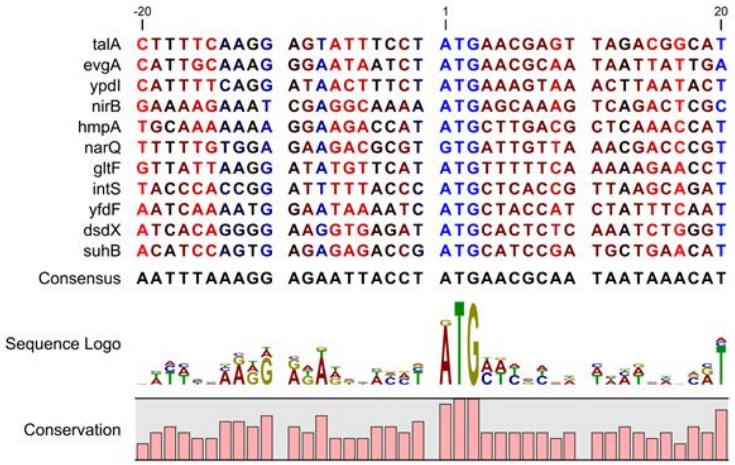
Bioinformatics'
methods
development

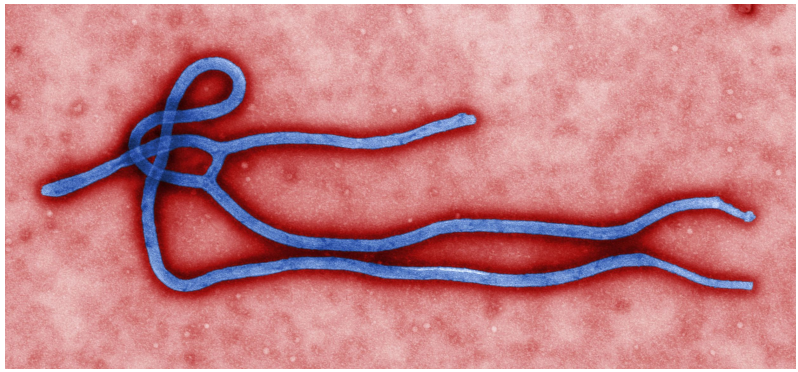
Processing
biological
data

Biology

Programming & bioinformatics

Goodness of your program is (mostly) defined by the **biological** question





Opinionated tips for programming

- Start small (*e.g.* not aligning 1000-genomes humans!) and one step at a time
- Don’t worry (about errors) (too much - testing is important, but...)
- Think! What...:
  - is the **biological question**?
  - is the **data**?
  - the program is supposed to do (methods, algorithms, ...)?
  - input (DNA-sequence? Set of RNA-seq data, names of plants, …)
  - can go wrong => then what (disk full, memory full, bad methods, too little data, ...)?
- Learn to **save your code** (naming, locations, even something like git)

Caveats

- **Everything** changes...
  - Data (WXS => WGS => WGBS; RNA-seq, …; HG37 vs. HG38...)
  - Methods (bowtie => bowtie2 => bwa mem => minimap2 => …)
  - Links go stale (404 Not Found)
  - Python 2.7 => 3.7+
  - **Python-libraries (Standard library, Numpy, Biopython, ...)**
  - Operating systems / platforms
  - System libraries

=> Do not get stuck with the old unless absolutely necessary, but don’t worry too much about newest trends!

Warmups...

More handy commands & constructs in Python

- Python core language has **a lot** of useful commands to make programming easier and simpler
- Same is true with libraries (this course)
- Beginner's class did not have time to go to many of the commands / constructs
- Thus, few more for your toolbox (read the documentation!)

List comprehensions

- “List comprehensions provide a concise way to create lists. Common applications are to make new lists where each element is the result of some operations applied to each member of another sequence or iterable, or to create a subsequence of those elements that satisfy a certain condition.” (*)

E.g.

```
mylist = [1, 2, 3, 5, 7, 11]
double = []
for number in mylist:
    double.append(2*number)
```

vs.

```
double = [ 2*number for number in mylist ]
```

(*) https://docs.python.org/3/tutorial/datastructures.html (5.1.3.)

with-statement / contex manager(...)

- Opening and especially closing files is a bit laborous in Python

- With-statement creates an “environment” that handles closing automatically

E.g:

```
file_handle = open(“expression_data.csv”, “r”)
# … work with the file
file_handle.close()
print(“All done!”)
```

vs.

```
with open(“expression_data.csv”, “r”) as file_handle:
    # … work with the file
print(“All done!”)
```

Errors and Exceptions 1/2

- Three(?) kinds of errrors:
  - syntax error (*e.g.* missing end parenthesis => program does not even start
  - runtime error (*e.g.* division by zero) => program crashes unintentionally
  - logical error (*e.g.* program calculates wrong thing) => bad science

- Which is the worst?

- Runtime errors (exceptions) can be handeled programmatically (to an extent) using `try` / `except`-statements (https://docs.python.org/3/tutorial/errors.html)

- Different exceptions can be handled separately (https://docs.python.org/3/library/exceptions.html#bltin-exceptions)

Errors and Exceptions 2/2

E.g.

```
mylist = [1,2,3,5,7,0,11]
inverse = [ 1/item for item in mylist ] # you know what will happen…
```

vs.

```
mylist = [1,2,3,5,7,0,11]
try:
    inverse = [ 1/item for item in mylist ]
except:
    print(“Sorry, there was a problem with the list”)
```

BUT! Be very, very careful when using exceptions! (Why?)

More on loops: `break`, `continue` and `pass` 1/2

- `break`-, `continue`- and `pass`-statements work inside `for`- and `while`-loops for control flow shortcuts (*)

- `break` will jump out of the innermost loop whereas `continue` will jump to the next iteration of the loop immediately

E.g.

```
for i in range(100):
    if i > 50:
        print(i)
        break
# what gets printed?
```

(*) https://docs.python.org/3/tutorial/controlflow.html (4.4)

More on loops: `break`, `continue` and `pass` 2/2

E.g.

```
for i in range(100):
    if i > 50:
        print(i)
        continue
    print(i*i)
# what gets printed?
```

- Beware! Sometimes `break` / `continue` obscures control flow - it is usually better to rethink why to loop in the first place

- The `pass`-statement does nothing. It can be used when a statement is required syntactically (rearly used)

Odds and sods

- There are many, many, many more (smallish and not so smallish) things in Python that make programmer's life easier:
  - enumerate-function => returns counter/index for *e.g.* list items along the item itself
  - zip-function => iterates in lock-step through lists and other iterables
  - print-statement accepts "file="-argument...
  - set-data structure => supports all mathematical set operations => easy to find *e.g.* differences
  - …

  - Standard library also provides all kinds of tools like:
    - Data presistence
    - Concurrent execution
    - … (more on libraries later)

=> study Python universe & practise!

Warm-up exercises

1) Get seqence-lenghts from a FASTA(*)-file (use “SH1_prots.fasta” - file) and print the shortest and the longest lengths.

2) Make file containing protein sequences (e.g. “my_sequences.txt” / one sequence per line) to be a proper multiFASTA-file.

(*) https://en.wikipedia.org/wiki/FASTA_format

Biology is **messy**

=>

data is messy

=>

do not panic => think!

https://en.wikipedia.org/wiki/KISS_principle

Exercise / tables => Pure version

Make a **pure**(*) Python-program(**) to read file “experiment_table_1_1000_first.csv” and multiply columns “treatment_2” and “treatment_12” together per value and list then the original columns “treatment_2” and “treatment_12” and the result in a new file.

(*) **pure** == just basic Python statements, no libraries needed or used.

(**) let’s call it e.g “column_multiplier_pure_python” for later use