Python for Biosciences -
return of the Janne

janne.ravantti@helsinki.fi
Slides & example data:

Ask questions!


& use search engines + stackoverflow, but be careful out there!
Goals for this time:

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

2) Pointers to some useful libraries

3) Practise Python programming
Today

1. Reflections from the last time
2. Python-programming warm up
3. Anaconda Distribution
4. Python as an Integration language
5. Libraries...
   5.1. Numpy & Scipy
   5.2. Matplotlib
   5.3. Pandas
   5.4. Biopython
   5.5. ...
6. Recap
Random thoughts about course this far...

- Too little time! Learning to program would need more than two days...
  
  =>

- Exercises are/were a bit too hard for many

- Jupyter notebook is rough on(in? at?) the edges

- Programming / problem solving with computer would need also little understanding of the operating system / programming environment (Linux, bash, ...)

- However, you have now started to program and have more than enough to keep going!
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  | Bioinformatics’ methods development  | Processing biological data  | Biology
Statistics | Mathematics |
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!
Warm-up exercises

1) Get sequence-lengths 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!

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
Python environments & libraries

- “Python applications will often use packages and modules that don’t come as part of the standard library.“ ([https://docs.python.org/3/tutorial/venv.html](https://docs.python.org/3/tutorial/venv.html))

- They can bee magical: [https://xkcd.com/353/](https://xkcd.com/353/)

- Or they can lead to madness: [https://xkcd.com/1987/](https://xkcd.com/1987/)
Data Science Handbook

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

https://stackoverflow.com/questions/40557910/plt-plot-meaning-of-0-and-1
Importing libraries to use 1/2

- The Python standard library (https://docs.python.org/3/library/index.html) includes many, many, many useful tools - use them, if you can!
  - => everything changes, standard library slower and with the language itself

- “import” statement brings additional tools/functions to programs to use

- There are several ways to use “import” (see e.g. https://stackoverflow.com/questions/9916878/importing-modules-in-python-best-practice)
Importing libraries to use 2/2

My recommendation, use either (e.g. importing pandas library):

```python
import pandas

my_table = pandas.read_csv("mydata.txt")

or

import pandas as pd

my_table = pd.read_csv("mydata.txt")
```
Library exercise - standard library

Make a Python-program that changes pair-ended reads from given FASTQ-file (use “my_reads.fq.gz”) to single-reads(**) and prints the modified file to a new file.

Notice the file type! You’ll need a little help from the standard library...

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

(**) basically, just make new unique read names
Anaconda Distribution

https://www.anaconda.com/what-is-anaconda/

“Easily install 1,400+ data science packages for Python/R and manage your packages, dependencies, and environments - all with the single click of a button(*). Free and open source.”

(*) or you can use command line
Anaconda Distribution

https://www.anaconda.com/what-is-anaconda/

“Easily install 1,400+ data science packages for Python/R and manage your packages, dependencies, and environments - all with the single click of a button(*). Free and open source.”

(*) or you can use command line

=> (IMHO) the easiest way currently to install and manage Python-environments e.g. to your own computer(s), clusters or in CSC’s machines
Anaconda Distribution

https://docs.anaconda.com/anaconda/

=> https://conda.io/docs/_downloads/conda-cheatsheet.pdf

See also:
https://stackoverflow.com/questions/42309333/explanation-of-different-conda-channels
Channels

**bioconda** provides also commonly used bioinformatic’s tools (bwa, samtools, …) with properly maintained library dependencies.

[https://bioconda.github.io/](https://bioconda.github.io/)

**conda-forge** is “A community led collection of recipes, build infrastructure and distributions for the conda package manager.”

[https://conda-forge.org/](https://conda-forge.org/)

Let’s test!
Some useful libraries for bio- & data sciences

https://www.numpy.org/

https://www.scipy.org/

https://matplotlib.org/

https://biopython.org/

https://pandas.pydata.org/

https://seaborn.pydata.org/

https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1004867
Python as an Integration language

Python-programs can be used as a “glue” e.g.:  

1)  Data => Python => command line program => Python => results  

2)  Python => (Data + command line programs) => results  

3)  Python => R => results(*)  

4)  “Data processing pipeline” (Python + R + command line + CSC-queues +...) => results  

(*) [https://warwick.ac.uk/fac/sci/moac/people/students/peter_cock/python/lin_reg](https://warwick.ac.uk/fac/sci/moac/people/students/peter_cock/python/lin_reg)
import subprocess

out = subprocess.check_output(['ls -l'], encoding='UTF-8', shell=True)

for l in out:
    print(l, end='')

Let's try!

Windows, use "dir"
Exercise

List all files from the jupyter-directory and find the largest in size.
Python ⇔ command line

Most Unix/Linux-commands accept arguments at command line (e.g. “ls -l work” => lists only files in “work”-directory)

We can do the same thing in Python => `sys.argv`

Caveat: this works easily only from the command line!

Let’s try:
Python ↔ command line (Linux / Mac only…)

1) Open new text file
2) Select language => Python

3) Make a Python-program to check disk usage at home directory (use Unix-command “du -sh” - more info “man du” from command line)

4) Save file as a real Python-program e.g. as “check_du.py”

5) Open (new) terminal
6) Run the program from command line python check_du.py
Python ⇔ command line

Run the program from command line “python check_du.py”

What if we want to find out disk usage of some particular directory only?

=>

Use `sys.argv[]`
import sys

print("Command line arguments:", sys.argv)

=> list of arguments starting from the program name
(https://docs.python.org/3/library/sys.html => sys.argv)

Exercise: modify check_du.py - program to accept a directory name as an argument
Closer look into libraries
Numpy

https://docs.scipy.org/doc/numpy-1.15.1/user/whatisnumpy.html

=> “NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.”

https://docs.scipy.org/doc/numpy-1.15.1/user/quickstart.html (N.B. indexing!)
Numpy arrays

- The only(?) thing with Numpy arrays is to be careful with the indexing
- **But!** the indexing makes a lot of sense and helps making coding cleaner

*E.g.:

```python
import numpy as np
a = np.arange(15).reshape(3,5) # two-dimensional table
a[:,2]                      # get a row
a[a>5]                      # get values that fulfill condition
```

[https://python4bioinformaticsblog.wordpress.com/index/python-bits/numpy/](https://python4bioinformaticsblog.wordpress.com/index/python-bits/numpy/)
[https://docs.scipy.org/doc/numpy/user/quickstart.html](https://docs.scipy.org/doc/numpy/user/quickstart.html)
Exercise / tables #2 /3 => Numpy

Modify your previous Python-program to use Numpy-library 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 values and the result.

Hints (Google): read numpy csv => which numpy method to use?

how to access columns in numpy => syntax for numpy arrays?
Scipy

“SciPy is a collection of mathematical algorithms and convenience functions built on the Numpy extension of Python. It adds significant power to the interactive Python session by providing the user with high-level commands and classes for manipulating and visualizing data. With SciPy an interactive Python session becomes a data-processing and system-prototyping environment rivaling systems such as MATLAB, IDL, Octave, R-Lab, and SciLab.”(*)

(*) https://docs.scipy.org/doc/scipy/reference/tutorial/general.html
Matplotlib

“Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, and four graphical user interface toolkits.” https://matplotlib.org/

=> for simple(?) plot pyplot(*) is often just fine:

```python
import matplotlib.pyplot as plt
```

(*): https://matplotlib.org/tutorials/introductory/pyplot.html#sphx-glr-tutorials-introductory-pyplot-py
Matplot exercise

Plot the sum of columns 2, 12 of the experiment_table_2_1000_first.csv.

*To see your plot in notebook, add the following in the beginning of your notebook:* 

```python
%matplotlib inline
```
Scipy & Matplotlib exercise

Modify your previous Python/Numpy-program to use Scipy-library calculate and draw linear regression between columns “treatment_2” and “treatment_12” from “experiment_table_1_1000_first.csv”

You can modify code from https://scipy-cookbook.readthedocs.io/items/LinearRegression.html, but note that the example has a lots of extra code! Use only the relevant parts...
Seaborn

“Seaborn builds on top of Matplotlib and introduces additional plot types. It also makes your traditional Matplotlib plots look a bit prettier.”


=> if you have time, compare resulting plots from your column_multiplier_pure_python - plot-program with Matplotlib and Seaborn
Seaborn

%matplotlib inline

import seaborn as sns

sns.set()

tips = sns.load_dataset("tips")

sns.relplot(x="total_bill", y="tip", col="time", hue="smoker", style="smoker", size="size",data=tips);
Pandas

“pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language” (https://pandas.pydata.org/)

https://pandas.pydata.org/pandas-docs/stable/10min.html

https://pandas.pydata.org/pandas-docs/stable/comparison_with_r.html

Pandas basic datatypes

- **Series** ≡ Numpy arrays with additional index
- **DataFrame** ≡ Numpy array + dictionary-type access

=> several methods to access and view data:
  - `df.info()`
  - `df.describe()`
  - ...

Missing values 1/2

- Always use **NaNs (Not a Number)** as missing values (WHY?)

```python
import numpy as np

np.nan == np.nan # (False!)
```

- Use `".replace(..., np.nan)"` to replace "bad" values with NaNs =>
  ```python
df = df.replace(-1, np.nan)
```

- You can e.g. remove rows having NaNs with `.dropna()`-method
Missing values 2/2

- You can *impute* (replace missing values with reasonable (?) guesses)

  *E.g.*

  ```python
  df_inputed = df.fillna(df.mean())
  ```

  (Machine learning library scikit-learn has more sophisticated methods, *e.g.*
  ```python
  df_imputed = SimpleImputer(missing_values=np.nan, strategy='mean')
  ```
Exercise

Modify your previous Python-program to use **Pandas**-library to read files "experiment_table_1_1000_first.csv" and "experiment_table_2_1000_first.csv" and then multiply columns “treatment_11” from both tables together. Print the results to a new csv-file.
Exercise

- Make a Python-program that reads a multi-FASTA-file, cleans up the header line to have only ID & gene-name and prints headers and sequences to standard output as an multi-FASTA-file again:

```python
>lcl|NC_007217.1_prot_YP_271858.1_1 [gene=HPSH1_gp01] [protein=ORF 1] [protein_id=YP_271858.1] [location=164..421]
=>

> YP_271858.1_##HPSH1_gp01
```

Tips: you can use file SH1_prots.fasta for the exercise
Biopython - introduction

- An Open Bioinformatics Foundation project
  - [https://www.open-bio.org/wiki/Projects](https://www.open-bio.org/wiki/Projects)
  - The idea is to provide common programming tools for various languages, including Python

- [http://biopython.org](http://biopython.org)
- [http://biopython.org/DIST/docs/tutorial/Tutorial.html](http://biopython.org/DIST/docs/tutorial/Tutorial.html)

- Can be called in Python by:
  ```python
  import Bio
  or specific sub-library e.g.
  from Bio import SeqIO
  import Bio.SeqIO
  # if import fails, install biopython-library
  ```
Biopython - capabilities

- [http://biopython.org/DIST/docs/tutorial/Tutorial.html#htoc2](http://biopython.org/DIST/docs/tutorial/Tutorial.html#htoc2)

- Mainly: dealing with biological sequences (DNA / RNA / proteins)

- *E.g.* nice ways to change sequence formats from command line:

  ```python
  import sys
  from Bio import SeqIO
  SeqIO.convert(sys.argv[1], "fasta", sys.argv[2], "clustal")
  ```

*Remember:* `sys.argv[]` takes filenames from a command line
(Bio)python - caveats

- Largish project based on volunteers
  - some parts might break ("API changes")
  - some parts might get much, much better

- Sometimes (Bio)python is not the best solution (hammer vs. nail)
  - sequences are strings => easy to manipulate with Python itself
  - other libraries exist (numpy, pandas, …)
    - e.g. data in tables, csv-files, …
  - other tools exist (e.g. EMBOSS)

=> learn to use also Linux & command line tools (CSC has nice courses!)
Biopython - sequences

- Sequences are everywhere in bioinformatics
- Biopython has many, many, many ways to work with sequences
- Sequences are string-like objects, with some additional information
  - all Biopython’s sequences have **alphabet**
  - alphabet defines type of the sequence (DNA / Protein)
- Biologically relevant methods per sequence-type
  - *e.g.* `my_dna.reverse_complement(); my_protein.translate()`
- [http://biopython.org/DIST/docs/tutorial/Tutorial.html#htoc17](http://biopython.org/DIST/docs/tutorial/Tutorial.html#htoc17)
Biopython - Blast

- Blast is arguably the single most important program in bioinformatics

- BioPython supports both WWW and local Blast-searches

- [http://biopython.org/DIST/docs/tutorial/Tutorial.html#htoc87](http://biopython.org/DIST/docs/tutorial/Tutorial.html#htoc87)

- Caveats
  - Blast has multitude of options - you need to understand them too!
  - Parsing Blast output is a bit complicated => see [http://biopython.org/DIST/docs/tutorial/Tutorial.html#fig:blastrecord](http://biopython.org/DIST/docs/tutorial/Tutorial.html#fig:blastrecord)
Biopython - Entrez

- Entrez is an interface to NCBI’s databases such as PubMed and GenBank
- Biopython supports Entrez in similar manner to Blast (handles, XML-output)
- [http://biopython.org/DIST/docs/tutorial/Tutorial.html#htoc111](http://biopython.org/DIST/docs/tutorial/Tutorial.html#htoc111)
- The output parsing can be confusing for a beginner...
Entrez - simple example

import Bio.Entrez
import Bio.SeqIO

Bio.Entrez.email = "janne.ravantti@helsinki.fi" # always tell who you are!

handle = Bio.Entrez.efetch(db="nucleotide", rettype="gb", retmode="text", id="NC_001421")

seq_record = Bio.SeqIO.read(handle, "gb")

handle.close()

print("Genbank ID:", seq_record.id)

print("Annotations:", seq_record.annotations)

print("Features:", seq_record.features)

print("Sekvenssi:", seq_record.seq)
Entrez - not so simple example...

import Bio.Entrez
import Bio.SeqIO

Bio.Entrez.email = "janne.ravantti@helsinki.fi" # always tell who you are!

handle = Bio.Entrez.esearch(db="pubmed", term="Ravantti")

record = Bio.Entrez.read(handle)

handle.close()

...

handle = Bio.Entrez.efetch(db="pubmed", retmode="xml", id="30375150")

rec = Bio.Entrez.read(handle)

...
Biopython - Entrez - exercise

- TT-Seq is a recent RNA-seq technique that maps a transient transcriptome.

- Make a Python-program that will find all TT-seq articles in Pubmed and prints how many there are and then print each article’s authors lastnames

*Do not get discouraged by the messy data! Use `type-function` to dissect the records and use appropriate keys/indices to dig deeper...*
File handling exercises

The problem: we have a directory (e.g. “example_data/sequences”) full of files that are either protein sequences, nucleotide/DNA sequences or ... “stuff”. Proper sequences are in FASTA-format.

1) Make a Python program that finds out which file is which

2) Modify your program that it copies files to new directories (e.g. “protein/”, “dna/” and “other/”)

3) Make a program that changes all headers to something unique for FASTA-files
Remember!

SAVE YOUR WORK FREQUENTLY!
Recap

● Python is well-suited for doing bioinformatics
  ○ easy(?) to learn
  ○ widely available
  ○ good standard library ("everything & kitchen sink!")
  ○ good / stable external libraries
  ○ performant with e.g. numpy

● However, things change, so plan accordingly
Background:

It is often useful to compare sets of sequences (genes of species chromosomes, ORFs of bacterial species, LINE-1-elements, contigs, ...) against each other and find e.g. the most similar(?) ones between the sets.

The most similar sequences can e.g. tell something about evolution of the species or point out, if there is a group of genes responsible for pathogenicity (i.e. genes appearing only in pathogenic strain).
Final Project 2/4

So, make a program that:

1) gets two sets of sequences in multi-fasta format

2) reports the most similar sequences between sets
Final Project 3/4

The current description is quite dense(?), so you might want to define subtasks and think how to program the following tasks:

- dealing with the errors & checking input
- definition of the comparison (similarity vs. identity vs. partial match (*)
- reporting format
  - scores only?
  - alignments?
  - visualization - genome diagrams and/or clustering?
- How to get sequences - download and/or read from the disk?
- REMEMBER TO ALSO DOCUMENT YOUR WORK!

(*) https://en.wikipedia.org/wiki/Sequence_alignment
Final Project 4/4

- See course page for
  - project description
  - deadlines
  - grading
  - example data (e.g. “JR1_nuc.fasta” & “SH1_nuc.fasta”)
  - documentation guidelines

- Ask questions and/or help!
THANK YOU!

janne.ravantti@helsinki.fi