Bioinformatics Resources

Lecture & Exercises
Prof. B. Rost, Dr. L. Richter, J. Reeb
Institut für Informatik I12
Bioinformatics Resources

- Organization
- Schedule
- Overview
Organization

- **Lecture:** Friday 9-12, i.e. 9.30-11.45 o’clock
  10 - 15 min break in between
  Room 00.13.009A

- **Exercise:** Monday 14-16 o’clock room
  00.08.038, starting Mon, May 8th
  Friday 13-15 o’clock room 01.09.014
  starting Fri, May 5th
Team Behind the Course

BioinfRes SoSe 17
## Preliminary Schedule

<table>
<thead>
<tr>
<th>Date</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apr. 28th</td>
<td>Intro, General Overview (1. sh.)</td>
</tr>
<tr>
<td>May 5th</td>
<td>Sequence Databases (2. sh.)</td>
</tr>
<tr>
<td>May 12th</td>
<td>Sequence Databases (3. sh.)</td>
</tr>
<tr>
<td>May 19th</td>
<td>Structure Databases (4. sh.)</td>
</tr>
<tr>
<td>May 26th</td>
<td>To be discussed w students</td>
</tr>
<tr>
<td>Jun 2nd</td>
<td>SQL (5. sh.)</td>
</tr>
<tr>
<td>Jun 9th</td>
<td>SQL, NoSql (6. sh)</td>
</tr>
<tr>
<td>Jun 16th</td>
<td>To be discussed w students</td>
</tr>
<tr>
<td>Jun 23rd</td>
<td>NoSql 2 (7.sh.)</td>
</tr>
<tr>
<td>Jun 30th</td>
<td>MongoDB, JavaScript (8.sh.)</td>
</tr>
<tr>
<td>Jul 7th</td>
<td>Node.js Applications (9.sh.)</td>
</tr>
<tr>
<td>Jul 14th</td>
<td>PredictProtein</td>
</tr>
<tr>
<td>Jul 21st</td>
<td>Wrap Up, Q&amp;A</td>
</tr>
<tr>
<td>Jul 28th</td>
<td>Exam</td>
</tr>
</tbody>
</table>

* These exercises can earn you a bonus
Schedule Details

- If there is no lecture the exercise takes place one week later
- Exercise sheets are published on Fridays and discussed Fri/Mon the week after
- Last sheet/exercise: Jul 7th, Fri/Mon 14th/17th
- Exam: (working date): July 28th, to be discussed with the audience, room yet undefined
Overview

- Lecture is still quite new – considered beta 2
- Third iteration
- Experience driven syllabus – subject to change
- Depending on the progress in the lecture single topics could be added or dropped
- The sequence of topics might be shuffled
- Hybrid nature: presentation of existing resources are blended with back- and front-end technology
Exercises

- Exercises help to convert knowledge into a skill
- Practical application of topics covered in the lecture
- Active exploration of bioinformatics resources
- Implementing various aspects of a bioinformatics resource
- Use Python/Biopython as common platform
- (Explore visualizations with JavaScript)
Terms and Meanings

- What does “resource” actually mean?
- A Google query about “Bioinformatics Resource” yields about 20 Mio hits
- Falls roughly into three categories:
  - databases
  - tools
  - service centers
Working on a Definition

- A collection of information which is useful to do research in the area of life sciences/computational biology
- Contains the information itself
- Provides appropriate interfaces to access the information
- (May provide) Tools for interactive data analysis
Genbank* / NCBI

- NIH genetic sequence database
- Annotated collection of all publicly available DNA sequences
- Part of the International Nucleotide Database Collaboration together with DNA DataBank of Japan (DDBJ) and the European Molecular Biology Laboratory (EMBL)
- Genbank flat file format

Genbank II

- New release every 2 months
- Retrievable via FTP from the NCBI website
- Current release is 219.0, April, 2017
- 231,824,951,552 bases from 200,877,884 sequences in Genbank; 2,035,032,639,80 from 451,840,147 sequences in WGS, Apr 2017
- (211,423,912,047 bases from 191,739,511 reported sequences, Apr 2016)
- (187,893,826,750 bases from 181,336,445 reported sequences, Feb 2015)
Genbank III

- Three main divisions: CoreNucleotide, dbEST, dbGSS
- Querying over Entrez Nucleotide
- Interactive BLAST analysis with user sequences
- Programmatic access via NCBI e-utils
Swissprot*

- Official name: UniProtKB/Swiss-Prot
- History
- Current release: 2017_04
- 554,241 (Apr. 2016:548,208) sequence entries comprising 198,410,167 residues from 251,858 references
- (550960 sequence entries, 195,282,524 residues abstracted from 235893 references last year)
- manually annotated


BioinfRes SoSe 17
Swissprot/Uniprot II

- manual annotation process
- standard operation procedure
- controlled vocabularies
- guidelines
- offered services: BLAST, Align, ID mapping
- associated services
Other Uniprot Services

- TrEMBL
- Proteomes
- UniRef
- UniParc
- programmatics access
PDB*

- History
- 129,367 structures, incl. 120,137 proteins by now
- (118,087 structures, incl. 115,169 proteins in 2016)
- (108124 structures, incl. 100450 proteins in 2015)
- PDB formats
- Data upload/validation
- Data dictionaries

*http://www.rcsb.org/pdb/statistics/holdings.do
PDB II

- Retrieval
- Programmatic access
- Visualization with the different views
- File format transitions: pdb and mmcif
SCOP/e

- Structural Classification of Proteins
- History, current version is SCOPe 2.05
- Changes in SCOPe
- Access
- Needed/recommended additional software
PFAM

- current version is 31.0, March 2017
- 16712 entries
- what is is about
- categories
- interactive use
- programmatic access
Prosite

- Prosite
  - current version 2017_04, Apr 12th, 2017
  - UniRule format and ProRule
  - access
  - typical use and interfaces
PubMed and discussion forums

- What is it for
- Search opportunities
- Linking to other information sources
- Search strategies
- A tour through various discussion forums
File Formats*

- Genbank/EMBL
- SWISSPROT
- PDB: mmCIF
- High Throughput NGS data:
  - BAM, SAM
  - VCF

* mostly integrated
File Formats

- Equivalence and transformations between different formats
- XML formats
- RDF formats
SQL

- SQL basics
- Data types
- Table creation and manipulation
- Join
- Select
SQL II

- Keys
- Indexes
- Performance influence of indexes
- Similarity search vs substrings
- Permissions
SQL III

- Transactions
- Setup, administration, backup
- Programmatic access
- mySQL, postgresQL
SQL IV

- General hints for database design
- Do’s and don’ts
- Normalization ultra light
No SQL

- Definitions of NoSQL
- Advantages / disadvantages
- Underlying theory
- Typical use cases
- Types of No-SQL database
- Query (languages)
No SQL Systems

- MongoDB
- (CouchDB)
- Neo4J
- Programmatic access
(Storing Facts)*

- Triple stores
- Data model
- RDF refresher
- Query language: sparql
- Examples

* optional, might be dropped
Programming Libraries

- Roadshow of programming libraries dedicated to bioinformatics:
  - (bioperl)
  - biopython
  - bioJS
  - Visualization
Graphical User Interfaces

- principles
- interaction modes
- modelling
- interaction modes
Graphical User Interfaces*

- Interactive user interfaces with JavaScript
- Language basics
- Programming model
- Client/server communication with json

* to be confirmed
JavaScript

- Libraries for data visualization/bioinformatics
- matplotlib
- bioJS
- D3
Client/Server Models

- Cgi
- Webservices
- Remote Procedure Calls / CORBA
- Security considerations
Authentication/Encryption

- authentication models
- communication encryption
- data/result encryption
- legal privacy issues
- data access models
Web Services I

- Types of web services
- Web service components
- Integration of web services in software
Web Services II

- Client side interfaces to web services
- Server side interfaces to web services
- Apache configuration for web services
- Required modules
- Configuration
- Performance
Bioinformatics Suites

- Where to find
- Installation/configuration
- Workflow systems: e.g. Taverna,.....
- EMBOSS, STADEN
- bio-.....
- .....
Selected Bioinformatics Suites

- Aquaria
- PredictProtein
- ....
Summary I

- Aim of this module:
  - Shape the concept of a bioinformatics resource
  - Become familiar with some of the most prominent examples out there
  - Get in touch with the underlying technology
  - Gather ideas and experience how to realize a new bioinformatics resource
Summary II

- Hands on (interaction) experience with existing experience
- Backend technology, i.e. various database models
- Frontend technology to realize the UI/ design rationales
- Communication models
Grading:

- Graded by a written exam 90/100 min
- Scheduled day xxx depends on:
  - available room
  - number of participants
- Exam admission: no admission limit
- With sufficient performance in the two marked exercises you can earn a bonus???
- The bonus applies only if you pass the exam
Exercises

- Exploration of available resources
- Simple to intermediate programming tasks
- Publication of the task in week x
- Presentation of solutions in week x+1
Exercises II

- 9 exercise sheets
- Work in groups of 2 for the bonus??
- Discussion with the audience
Exercises III

- Groups fixed for the bonus
- New sheets are published on Friday
- Submission is due on Friday morning for all groups
- Two slots for exercises
Questions & Answers
Programming Exercises

- We will use Python for our programming exercises
- Scripting language
- Basic understanding of Python should be sufficient to understand the presented code snippets
- Vivid community for support and development
Programming Exercises II

- Object oriented
- Good integration with database systems and web access
- Good integration with sophisticated data analysis tools like: numPy, sciPy, mathplotlib
- BioPython
- Interactive notes: jupyter and notebooks
Structure your research work

- Computational biology is data driven
- Results matter -> more results matter more
- Other than e.g. software development there is no final release version and all prior bugs/version are abandoned
- Appropriate documentation of the experiments to reconstruct the intermediate steps is important, otherwise you may with result01 - result1000 files
Our preferred Software Setup

- Anaconda
- iPython/Jupyter notebooks
Anaconda

- Python distribution ([https://www.continuum.io](https://www.continuum.io))
- Clever packet manager: conda
- Allows a complete installation including various configuration next to each other in the user space
- No privileges needed
- Your host system is not modified
- Works with Windows, OS X, Linux
Some snippets from the conda cheat sheet

- Use “conda create –n xxx biopython” to create a new environment xxx and install biopython
- Use “(source) activate xxx” to activate this environment in your shell
- Allows different versions of python to be installed at the same time
iPython/Jupyter

- [http://jupyter.org](http://jupyter.org)
- Supports many different languages, we use it for python
- Use conda to install the package (if not installed): conda install jupyter
- Easy start of notebook: jupyter notebook
Advantages of a Notebook

- Allows you a seamless integration of:
  - (rich) text
  - (live) code
  - (live) results
  - visualizations
- Tie together your analysis script, the results and an interpretation/discussion
- You can archive and share the notebooks easily
More Details about Notebooks

- Supports and records interactive analysis session
- Variables and values are persistent between cells
- Provides full power of Data Science tools via imported modules: Bio, Pandas, Numpy, Scipy
- Good data processing: Pandas
- Good statistics support: Numpy, Scipy
- Good visualization support: Matplotlib
Biopython

- Advantage of easier / more clear syntax than Perl
- Oriented to BioPerl
- Supports a lot of common bioinformatics file formats
- Supports access to online services like NCBI, Expasy ...
- More interfaces for bioinformatics software

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

BioinfRes SoSe 17
Biopython

- [http://biopython.org](http://biopython.org)

- If installed: “import Bio” loads it in your scripts


```python
from Bio.Seq import Seq

# create a sequence object
my_seq = Seq('CATGTAGACTAG')

# print out some details about it
print 'seq %s is %i bases long' % (my_seq, len(my_seq))
print 'reverse complement is %s' % my_seq.reverse_complement()
print 'protein translation is %s' % my_seq.translate()
```
Biopython

seq CATGTAGACTAG is 12 bases long
reverse complement is CTAGTCTACATG
protein translation is HVD*

taken from http://biopython.org/wiki/SeqIO:
from Bio import SeqIO
handle = open("example.fasta", "rU")
for record in SeqIO.parse(handle, "fasta") :
    print record.id
handle.close()

from Bio import SeqIO
record = SeqIO.read(open("single.fasta"), "fasta")
Dedicated Data Structures

- sequence (Seq): beside the sequence of residues it allows also to provide an Alphabet object -> kind of type safety for DNA and protein sequences
- Typical functions like complement(), reverse_complement()
Dedicated Data Structures

- Parsing functions for different sequence formats
- Parsing functions alignment formats know about the different components
- As well as respective output functions
- Different translation tables
- Various predefined alphabets
Python Basics (Version 3)

- https://docs.python.org/3/tutorial/
- https://py-tutorial-de.readthedocs.io/de/python-3.3/
- Good interactive handling, i.e. you can evolve and evaluate your code directly in a python shell
- Later you can include it in your script
Data Types (not complete)

- Basic data types:
  - numerical types comparable to Perl, C, Java
  - strings
  - boolean: True, False

- Complex collection types

- Functions are first class members i.e. you can assign a function to a variable and you can return a function for another function
Functions As Return Values

def genPower(power):
    def specPower(base):
        return base ** power
    return specPower

square = genPower(2)
cube = genPower(3)
square(5)   # 25
cube(5)     # 125
Important Syntax

- White space (tabs, spaces) and : are used to structure the code in blocks, similar to {} in other languages
- Same indentation == same block
- Blocks of if/for/while etc. statements are indented
- No semicolons needed
- Parentheses often not mandatory
- Usual control structures available
Loops

- while: needs a boolean condition
- for: needs a list to be iterated
- use range(x) to generate the list 0-x for x iterations
- break and continue analogous to C or Java

```python
for w in words:
    print (w, len(w))

# if you want to iterate by numbers you
# have to use range()
for i in range(len(a)):
    print i, a[i]
```
Functions and Arguments

- **Definition of functions:**

  ```python
def fib(n):  # write Fibonacci series up to n
    """Print a Fibonacci series up to n. """
    a, b = 0, 1
    while a < n:
        print a,
        a, b = b, a+b
  ```

- Arguments can be passed by name or position
- Arguments can have default values - optional in the call
- Default values are evaluated at the point of definition
Functions and Arguments

def parrot(voltage, state='a stiff', action='voom', type='Norwegian Blue'):
    print("-- This parrot wouldn't", action, end=' ')
    print("if you put", voltage, "volts through it.")
    print("-- Lovely plumage, the", type)
    print("-- It's", state, "!")

parrot(1000) # 1 positional argument
parrot(voltage=1000) # 1 keyword argument
parrot(voltage=1000000, action='VOOOOOM') # 2 keyword arguments
parrot(action='VOOOOOM', voltage=1000000) # 2 keyword arguments
parrot('a million', 'bereft of life', 'jump') # 3 positional arguments
parrot('a thousand', state='pushing up the daisies') # 1 positional, 1 keyword

taken from https://docs.python.org/3/tutorial/controlflow.html
Functions and Arguments

- even more arguments are possible:
  - *anyName: a tuple containing additional positional arguments
  - **anyDifferentName: a dictionary containing additional keyword arguments
  - *anyName must occur before **anyDifferentName

- if you need a stub doing nothing: pass
Useful Data Types

- **Sequence types:**
  - Supports easy check for an element
  - Mutable types: List, Bytearray
  - Immutable: String, Tuple

- **Slicing:** act on sub sets not only on single elements
Other Collection Types

- **Set**: every element exists only once
- **Dictionary**:
  - can store key/value pairs
  - key has to be immutable (hashable)
- **derived from Dictionary**:
  - DefaultDictionary
  - Counter
List Comprehension

- very powerful
- compact notation of looping of lists
  - needs:
    - substrate list
    - function/operator
  - yields:
    - derived list

BioinfRes SoSe 17
Example for List Comprehension

```python
r = range(10)
>>> print(r)
range(0, 10)

>>> [a for a in r]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

>>> [a*2 for a in r]
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]

>>> [a*2 for a in r if a%2 == 0]
[0, 4, 8, 12, 16]
```
Slicing

- `[]` operator for lists
- allows ranges: `from .... to
- `a = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]`
- `a[1:5]`
- `[1, 2, 3, 4]`
- analogous for Dictionaries and string lists