

# Python im Bioinformatiker-Alltag

Marcel Martin

Bioinformatik für Hochdurchsatztechnologien, TU Dortmund

6. Oktober 2011

## Worum gehts?

### Bioinformatik

Löst Probleme in Biologie oder Medizin

### Teilbereich Genominformatik

Analyse des Erbguts (Genom) von Lebewesen

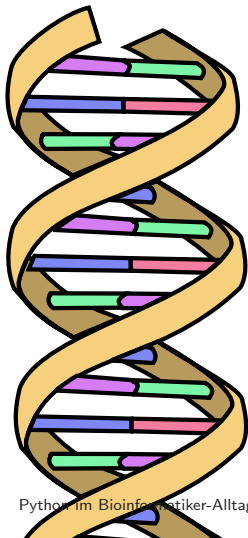
# Übersicht

- 1 Ein bisschen Biologie
- 2 Sequenzdaten und Datenformate
- 3 Wie wir Python einsetzen
- 4 Unsere Python-Software
- 5 Gedanken zu Python in der Bioinformatik

# Übersicht

- 1** Ein bisschen Biologie
- 2 Sequenzdaten und Datenformate
- 3 Wie wir Python einsetzen
- 4 Unsere Python-Software
- 5 Gedanken zu Python in der Bioinformatik

## Molekularbiologie



### DNA – Träger der Erbinformation

- Für Biologen: Zwei Kettenmoleküle aus vier **Basen** – **A**denin, **C**ytosin, **G**uanin, **T**hymिन
- Paarung: immer **A–T** und **C–G**
- Für Informatiker: String über dem Alphabet {A, C, G, T}
- Gen: Ein Abschnitt auf der DNA

PD by Forluvoft

## Was machen wir?

### Fragen

- Welche Gene sind in bestimmten Zelltypen aktiv und wie stark?

## Was machen wir?

### Fragen

- Welche Gene sind in bestimmten Zelltypen aktiv und wie stark?
- Wie unterscheidet sich Genaktivität in kranken und gesunden Zellen? Welche Gene sind „an“, welche „aus“?

## Was machen wir?

### Fragen

- Welche Gene sind in bestimmten Zelltypen aktiv und wie stark?
- Wie unterscheidet sich Genaktivität in kranken und gesunden Zellen? Welche Gene sind „an“, welche „aus“?
- Welche Mutationen führen zu Krebs?

## Was machen wir?

### Fragen

- Welche Gene sind in bestimmten Zelltypen aktiv und wie stark?
- Wie unterscheidet sich Genaktivität in kranken und gesunden Zellen? Welche Gene sind „an“, welche „aus“?
- Welche Mutationen führen zu Krebs?
- Welche Veränderungen in welchem Gen sind für eine erbliche Krankheit verantwortlich?

## Was machen wir?

### Fragen

- Welche Gene sind in bestimmten Zelltypen aktiv und wie stark?
- Wie unterscheidet sich Genaktivität in kranken und gesunden Zellen? Welche Gene sind „an“, welche „aus“?
- Welche Mutationen führen zu Krebs?
- Welche Veränderungen in welchem Gen sind für eine erbliche Krankheit verantwortlich?

⇒ Beantwortung mit Hochdurchsatz-DNA-Sequenzierung

## Hochdurchsatz-DNA-Sequenzierung

DNA-Fragmente aus Gewebeprobe (z. B. Blut)



Sequenziergerät



Kurze Zeichenketten bestehend aus A, C, G, T: **Reads**

## Technische Daten

### Illumina HiSeq

- 100 Basen pro Read
- 6 Milliarden Reads pro Durchlauf (14 Tage)
- 600 *Gigabasen* pro Durchlauf

## Technische Daten

### Illumina HiSeq

- 100 Basen pro Read
- 6 Milliarden Reads pro Durchlauf (14 Tage)
- 600 *Gigabasen* pro Durchlauf

### Zum Vergleich

Humangenom: 6 Gigabasen (in  $2 \times 23$  Chromosomen)

## Technische Daten

### Illumina HiSeq

- 100 Basen pro Read
- 6 Milliarden Reads pro Durchlauf (14 Tage)
- 600 *Gigabasen* pro Durchlauf

### Zum Vergleich

Humangenom: 6 Gigabasen (in  $2 \times 23$  Chromosomen)

### Gerätesoftware

Frontend: Vista; Compute-Server: Linux  
Software teilw. in Python

# Übersicht

- 1 Ein bisschen Biologie
- 2 Sequenzdaten und Datenformate**
- 3 Wie wir Python einsetzen
- 4 Unsere Python-Software
- 5 Gedanken zu Python in der Bioinformatik



## Was geschieht mit den Daten?

### Read Mapping (wenn Genom der Spezies bekannt)

- Positioniere Reads mit Textsuchalgorithmen auf dem Genom
- **Erlaube Unterschiede** – Sequenzierfehler oder Mutationen

### Finde Mutationen

Vergleiche bekannte Referenz mit Reads und liste abweichende Stellen auf

## Read Mapping

- Platziere Read auf Chromosom
- Erlaube: Zeichenveränderung, -löschung, -einfügung

### Alignment („-“: Lücke)

```
Read:          CT-ATCGCTTCTGTC
Chromosom:    ... CAGCCAGGCTGATCCCTT-TGTCGAGTAGGAC...
```

diff macht Ähnliches

## SAM-Dateiformat

Ausgabe der positionierten Reads im **SAM-Dateiformat**

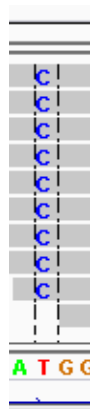
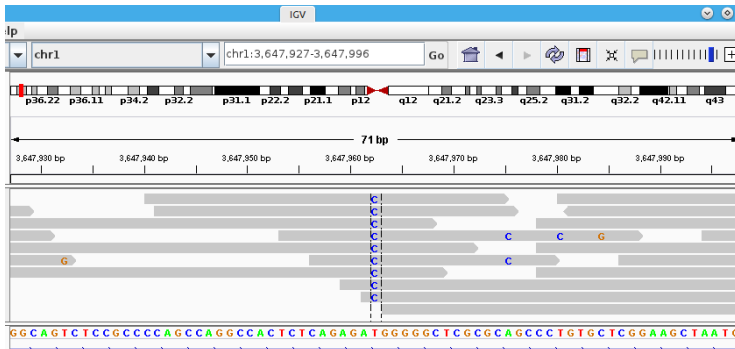
- 1 Readname
- 2 Chromosom und Position
- 3 Alignment
- 4 Sequenz und Qualitätswerte

Python-API für Zugriff auf SAM-Dateien: `pysam`<sup>1</sup> (nicht von uns)

---

<sup>1</sup><http://pysam.googlecode.com/>

# SAM-Viewer



# Übersicht

- 1 Ein bisschen Biologie
- 2 Sequenzdaten und Datenformate
- 3 Wie wir Python einsetzen**
- 4 Unsere Python-Software
- 5 Gedanken zu Python in der Bioinformatik

## Software-Arten

- Einmal-Skripte
- Software-Bibliothek
- Neue Algorithmen

## Einmal-Skripte

- Nur in einem Projekt interessant
- Wenn als Einzeiler möglich, dann in Bash, ansonsten in Python programmiert
- „quick and dirty“, aber: immer mit (kurzer) Doku

### Beispiele

- Konvertiere andere Sequenzformate zu FASTQ
- Importiere Mutationen in SQLite-Datenbank, führe ein SELECT aus, gib Ergebnis aus

## Bibliothek / Sammlung

### Anforderungen

- Aufnahme wenn mehrfach ( $\geq 2$ ) benötigt
- Hartkodierte Werte ok – aber sofort Kommandozeilenparameter hinzufügen wenn nötig
- Wenn ich im Code nachschauen muss → Doku verbessern
- Zur Zeit nur teilweise: Unit Tests

## Entwicklung neuer Algorithmen

### Beispiele

- Read-Mapping-Algorithmus für Sequenziertechnologie mit besonderen Fehlertypen
- Suffix-Array-Konstruktionsalgorithmus

### Ablauf

- Prototyp immer in Python → schnelle iterative Verbesserung
- Evaluation auf kleinen Testdaten
- Prototyp auf große Daten loslassen
- Langsam? → Parallelisieren (Cluster), C-Erweiterung programmieren oder abwarten

## Effizienz

### Python ist langsam! Oder?

- Python ist schnell und effizient – zu programmieren
- Rechenzeit ist billig, Arbeitszeit ist teuer

### Beispiel

Fremdes C++-Tool zum Kürzen von Reads in FASTQ-Datei –  
reimplementiert in reinem Python

→ Python war **doppelt so schnell**

Wahl der Algorithmen ist viel wichtiger als Wahl der  
Programmiersprache!

# Übersicht

- 1 Ein bisschen Biologie
- 2 Sequenzdaten und Datenformate
- 3 Wie wir Python einsetzen
- 4 Unsere Python-Software**
- 5 Gedanken zu Python in der Bioinformatik

## sqt – SeQuencing Tools

- Kommandozeilentools (Python- und Shell-Skripte)
- Python-Module (in Python und C)

MIT-Lizenz, <http://sqt.googlecode.com>

## sqt – SeQuencing Tools

- Kommandozeilentools (Python- und Shell-Skripte)
- Python-Module (in Python und C)
- Idee ähnlich wie Git: sqt-Binary plus externe Subcommands in beliebiger Programmiersprache (sogar Perl)

MIT-Lizenz, <http://sqt.googlecode.com>

## sqt – SeQuencing Tools

- Kommandozeilentools (Python- und Shell-Skripte)
- Python-Module (in Python und C)
- Idee ähnlich wie Git: sqt-Binary plus externe Subcommands in beliebiger Programmiersprache (sogar Perl)
- Momentan unvollständig

MIT-Lizenz, <http://sqt.googlecode.com>

## Beispiel: fastamutate

```
1 import sys
2 from sqt import seqio
3
4 def mutate(seq, rate=0.01):
5     # mutate and return
6
7     out = seqio.FastaWriter(sys.stdout, wrap=80)
8     for record in seqio.FastaReader(sys.argv[1]):
9         mutated = mutate(record.sequence)
10        out.write(record.name, mutated)
```

## Cutadapt

- DNA-Moleküle enthalten am Ende „Adaptoren“ (techn. Gründe)
- Wenn DNA-Fragment zu kurz, wird Adapter mitsequenziert

GAGTGGTTGGAGAGGAGTTGTTGGGAGTTTGTGTCCTGCTGAGACACGCA

- **cutadapt** schneidet Adapter fehlertolerant ab
- Füllt anscheinend eine „Marktlücke“
- Läuft mit Python 2.6 bis 3.2

MIT-Lizenz, <http://cutadapt.googlecode.com>

# Übersicht

- 1 Ein bisschen Biologie
- 2 Sequenzdaten und Datenformate
- 3 Wie wir Python einsetzen
- 4 Unsere Python-Software
- 5 Gedanken zu Python in der Bioinformatik**

## Python in der Bioinformatik – ja bitte

- Bioinformatik-Tools: Es scheint alles zu geben – aber schlecht (Abstürze, seltsame Ein- oder Ausgabeformate, verwaist, unsinnige oder keine Fehlermeldungen)
- Das wäre auch mit Python möglich – ist aber schwieriger! (Exceptions, argparse)
- Python-Code ist aufs Wesentliche reduziert – macht das Verstehen von Algorithmen einfacher

## Um was es ging

- Moderne Sequenziergeräten produzieren Massen von Daten
- Python lässt sich dennoch sinnvoll anwenden
- Python-Code muss nicht von Anfang an wartbar und schön sein
- Die Bioinformatik verträgt mehr Python – es würde Arbeit erleichtern und macht einfach Spaß!

## Quelltextkompatibel mit Python 2 und 3

- überall:

```
from __future__ import print_function, division
```

- 5–6 Stellen mit:

```
if sys.version_info[0] >= 3:  
(darin z.B. xrange = range)
```

- Größtes Problem: **str** vs. **bytes/bytearray** (Elemente sind Strings der Länge 1 oder ints)
- Wie erstellt man sinnvoll installierbare gemischte Pakete aus Python-2- und Python-3-Tools?