1 Block 1: Das erste `Perl` Programm – Datenausgabe auf dem Bildschirm

1.1 Lernziele

Ziel dieses Blocks ist es zu verstehen

- wie biologische Sequenzinformation in der Bioinformatik dargestellt wird;
- wie ein einfaches `Perl` Programm aufgebaut ist;
- einen vordefinierten Programmtext abzuschreiben, das Programm auszuführen und syntaktische Fehler zu beheben;
- was folgende Begriffe bedeuten und wie sie programmatisch umgesetzt werden:
  - Kommandozeilen-Interpretation;
  - Kommentar;
  - Variablendeklaration und Zuweisungsbefehl;
  - `print` und `exit` Befehl;
- einen vordefinierten Programmtext leicht abzuwandeln, zu speichern und auszuführen.

1.2 Einführung / Theorie

1.2.1 Sequenzdaten und Zeichenalphabete

Ein Grossteil der Datenverarbeitung in der Bioinformatik beschäftigt sich mit dem Speichern, der Ausgabe und Verarbeitung biologischer Sequenzen. Grundsätzlich sind zwei Typen von **Sequenzdaten** zu unterscheiden:

- *Nukleotidsequenzen* - für Repräsentierung von Genomen- und Genen (RNA und DNA);
- *Aminosäuresequenzen* – für die Repräsentierung von Proteinen;

Für die Darstellung dieser Daten mithilfe des Computers werden definierte **Zeichenalphabete** verwendet.

Für Nukleotidsequenzen:

Symbol	Bedeutung	Ursprung der Bezeichnung
G	G	Guanin
A	A	Adenin
T	T	Thymin
C	C	Cytosin
U*	U	Uracil
R	G oder A	puRin
Y	T oder C	pYrimidin
M	A oder C	aMino
K	G oder T	Keto
S	G oder C	Starke Wechselwirkung (3 H Brücken)
W	A oder T	Schwache Wechselwirkung (2 H Brücken)
H	A oder C oder T	nicht-G, H folgt G in Alphabet
B	G oder T oder C	nicht-A, B folgt A
V	G oder C oder A	nicht-T (nicht-U), V folgt U
D	G oder A oder T	nicht-C, D folgt C
N	G oder A oder T oder C	aNy

* Für Ribonukleotidsequenzen (RNAs)

Für Aminosäuresequenzen:

Aminosäure	Einzelzeichen-Code	Nukleotid-Triplet (5'-3')
Glycin	**G**	GGN

Alanin	**A**	GCN
Valin	**V**	GTN
Leucin	**L**	*YTN* (CTN und TTR)
Isoleucin	**I**	ATH
Prolin	**P**	CCN
Phenylalanin	**F**	TTY
Tyrosin	**Y**	TAY
Cystein	**C**	TGY
Methionin	**M**	ATG
Histidin	**H**	CAY
Lysin	**K**	AAR
Arginin	**R**	*MGN* (CGN und AGR)
Tryptophan	**W**	TGG
Serin	**S**	*WSN* (TCN und AGY)
Threonin	**T**	ACN
Aspartat	**D**	GAY
Glutamat	**E**	GAR
Asparagin	**N**	AAY
Glutamin	**Q**	CAR
Aspartat oder Asparagin	**B**	RAY
Glutamat oder Glutamin	**Z**	SAR
Terminator	**.**	*TRR* (TAR und TGA)
Unbekannt	**X**	NNN

Aus diesen Alphabeten können Zeichenfolgen zusammengesetzt werden: so genannte *Strings*. Diese Strings sind die elementaren Einheiten in der Repräsentierung biologischer Sequenzinformation.

1.2.2 Beispiele

Welche typischen Beispiele aus der Biologie, in denen Sequenzen als Zeichenketten repräsentiert werden, sind Ihnen bekannt?

Für Nukleotidsequenzen z.B.:

- Sequenzen von Gene und Genome;
- Erkennungsstellen für Restriktionsenzyme;
- DNA-Bindestellen von Transkriptionsfaktoren;
- PCR-Primer (Oligonukleotidsequenz);

Für Aminsäuresequenzen z.B.:

- Sequenzen von Proteinen;
- Proteinsequenzmotive;

1.3 Praxis

Anhand eines konkreten Programmbeispiels werden wir jetzt schrittweise erlernen wie mithilfe der Programmiersprache `Perl` biologische Sequenzinformation repräsentiert und manipuliert werden kann. Der `Perl` Interpreter führt Programme im Wesentlichen *zeilenweise* aus. Deswegen lässt sich der Programmsablauf auf diese Weise gut nachvollziehen. Für jede Zeile wird die Bedeutung der Programmsprache, -die so genannten *Syntax*-, erläutert.

1.3.1 Zeilenweises Erläutern des Programmbeispiels `Seq_Aus.pl`

Betrachten wir folgendes `Perl` Programm, das eine Nukleotidsequenz speichert und auf dem Bildschirm ausgibt (die Zeilennummerierung, - in gelb-, in der linken Spalte ist nicht Teil des Programmtextes:

```
 1  #!/usr/bin/perl -w
 2  # Dieses Programm speichert eine DNA Sequenz in einer Variablen und gibt sie
    auf den Bildschirm aus.
 3  # Speichern einer Sequenz in einer Variablen
 4  $DNA_Sequenz = 'ACGGTGCTGGACTGATTAGCGTATATAGC';
 5  # Ausgabe der Sequenz auf den Bildschirm
 6  print $DNA_Sequenz;
 7  # Ausgabe einer Leerzeile auf den Bildschirm
 8  print "\n";
 9  # Beenden des Programms
10  exit;
```

Wir werden nun zeilenweise durch den Programmtext gehen und die einzelnen Schritte erläutern. Interpretierte Befehlszeilen innerhalb eines `Perl` Programms enden immer mit einem Semikolon `;` gefolgt von einem Zeilenvorschub.

1.3.1.1 Kommandozeilen-Interpretation `#!`

Die erste Zeile

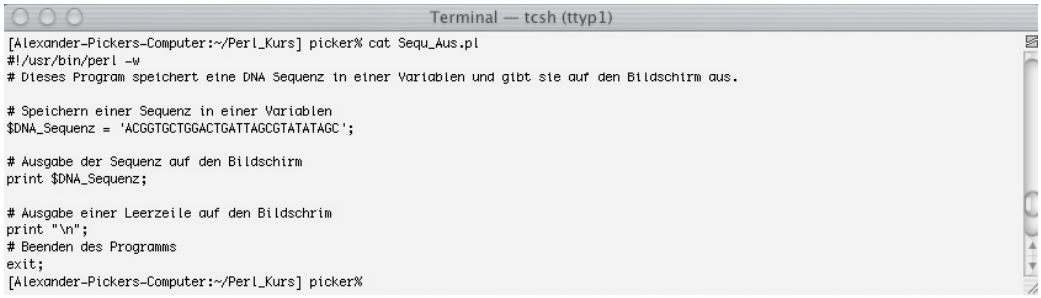
```
#!/usr/bin/perl -w
```

ist die so genannte *Kommandozeilen-Interpretation*. Computern, die mit den Betriebssystemen Linux oder Unix laufen, teilt diese Zeile mit, dass es sich um ein Perl Programm handelt. `/usr/bin/perl` teilt dem Betriebssystem mit, dass `Seq_Aus.pl` mit dem Programm *Perl*, dass sich im Unterverzeichnis `bin` im Verzeichnis `usr` befindet ausgeführt werden muss. Der Begriff *Perl* steht also nicht nur für eine Programmiersprache sondern auch für ein Computerprogramm, den so genannten *Perl-Interpreter*, der Programme die in Perl geschrieben sind ausführen kann, so wie unser Beispiel `Seq_Aus.pl`.

Wenn diese Zeile Im Programmtext enthalten ist kann das Programm direkt mit dem Befehl

```
% Seq_Aus.pl
```

von der Kommandozeile aus gestartet / ausgeführt werden. Die Kommandozeile ist das textuelle *Interface* mit der Betriebssystemebene des Computers:



1.3.1.2 Kommentare `#`

Die zweite

```
# Dieses Programm speichert eine DNA Sequenz in einer Variablen und gibt sie
auf den Bildschirm aus.
```

und dritte Zeile

```
# Speichern einer Sequenz in einer Variablen
```

sind *Kommentare*. Zeilen, die mit dem `#` Zeichen beginnen werden von `Perl` ignoriert. Kommentare dienen dazu den Programmablauf innerhalb des Programmtextes zu erläutern, zum eigenen Verständnis des Programmierers und anderer. Desweiteren sind sie ein einfaches aber weit verbreitetes Hilfsmittel zum Erarbeiten von Programmtexten.

1.3.1.3 Variablendeklaration `$x;` und Zuweisungsbefehl `$x=y;`

Die vierte Zeile des Programmtextes

```
$DNA_Sequenz = 'ACGGTGCTGGACTGATTAGCGTATATAGC';
```

enthält einen *Befehl* der aus zwei Teilen besteht: einer *Variablendeklaration* und die *Zuweisung eines Wertes an* diese *Variable*.

Mit

```
$DNA_Sequenz # Variablendeklaration
```

wird dem Perl Interpreter mitgeteilt dass eine *Variable* mit dem Namen `DNA_Sequenz` bereitgestellt werden soll. Die Variable ist vom Typ eine *skalare* Variable. Das bedeutet, dass sie die Variable nur *einen* Wert enthält, in unserem Fall die Zeichenkette `ACGGTGCTGGACTGATTAGCGTATATAGC`. Alle skalaren Variablen beginnen mit dem `$` Zeichen.

Variablennamen sollen sinnvoll soll, also den Inhalt der Variablen wiedergeben und dürfen Gross- und Kleinbuchstaben, Zahlen und den Unterstrich `_` enthalten aber nicht mit einer Zahl beginnen.

Mit

```
$DNA_Sequenz = 'ACGGTGCTGGACTGATTAGCGTATATAGC'; # Wertzuweisung
```

wird dem `Perl` Interpreter mitgeteilt, der Variablen den Wert `ACGGTGCTGGACTGATTAGCGTATATAGC` zuzuweisen. Der Wert ist in diesem fall vom Typ *String*, also ein Zeichen oder eine Zeichenkette. Das = Zeichen in dem Befehl ist ein Operator, der so genannte *Zuweisungsoperator*. Nach einer solchen Zuweisung kann man den Variablennamen `DNA_Sequenz` verwenden um auf ihren Wert `ACGGTGCTGGACTGATTAGCGTATATAGC` zuzugreifen.

Eine alternative, *explizite* Schreibweise der Zeile wäre:

```
$DNA_Sequenz; # Deklaration
$DNA_Sequenz = 'ACGGTGCTGGACTGATTAGCGTATATAGC'; # Zuweisung
```

mit der Deklaration und Zuweisung in zwei separaten Zeilen. Oft ist es sinnvoll am Anfang eines Programmtextes alle relevanten Variablen zu deklarieren und ihre Verwendung durch Kommentare zu dokumentieren. Das hält den Programmtext übersichtlich und ermöglicht es ihn an einer zentralen Stelle zu modifizieren.

Andererseits sind insbesondere routinierte `Perl` Programmierer notorisch schreibfaul. Deswegen ist es wichtig zu verstehen wie stark man die `Perl` Syntax verkürzen kann, um fremde Programmtexte lesen, verstehen und modifizieren zu können.

1.3.1.4 Der `print` Befehl

Die vierte

```
print $DNA_Sequenz;
```

und achte Zeile

```
print "\n";
```

des Programmtextes sind so genannte `print` Befehle. Dieser Befehl produziert eine Ausgabe auf dem Computerbildschirm: `print $DNA_Sequenz;` greift auf den Wert der Variablen `$DNA_Sequenz` zu und gibt ihn aus und `print "\n";` eine Leerzeile.

1.3.1.5 Der `exit` Befehl

Die letzte Zeile des Programmtextes

```
exit;
```

Enthält einen so genannten `exit` Befehl. Dieser Befehl teilt dem `Perl` Interpreter mit, das Programm zu beenden. In `Perl` ist der `exit` Befehl kein Muss. Manchmal ist es sinnvoll `exit` Befehle auch an anderen Stellen eines Programms einzubauen, wenn man den Programmablauf bewusst beenden will, zum Beispiel bei fehlerhafter Benutzung.

1.3.2 Übung

1.3.2.1 Aufgabe

- Schreiben sie den Programmtext von `Seq_Aus.pl` in ihrer Entwicklungsumgebung ab. Speichern sie das Programm unter dem Namen `Seq_Aus.pl` an einem Platz auf ihrem lokalen Computer und führen sie es aus. Beheben sie anhand der Vorlage mögliche syntaktische Fehler, die beim Abschreiben der Programmtextes aufgetreten sind.

- Modifizieren sie folgende Elemente von `Seq_Aus.pl` in einer Arbeits-Kopie der Originaldatei `Name_Aus.pl`:
  - Ändern sie den Variablennamen von `$DNA_Sequenz` zu `$Mein_Name`;
  - Ändern sie den Wert der Variablen, so dass ihr Vor- und Nachname zugewiesen wird;
  - Ändern sie den `print` Befehl, so dass

    ```
    Mein Name ist: Vorname Nachname
    ```

    auf dem Bildschirm ausgegeben wird;
  - Ändern sie die Kommentare in sinnvoller Weise.

1.3.2.2 Lösung

Programmtext von `Name_Aus.pl`.

```
#!/usr/bin/perl -w
# Dieses Program speichert einen Namen in einer Variablen und gibt ihn auf den
Bildschirm aus.

# Speichern des Namens in einer Variablen
$Mein_Name = 'Fritzchen Mueller';

# Ausgabe des Namens auf dem Bildschirm
print $Mein_Name;

# Ausgabe einer Leerzeile auf den Bildschrim
print "\n";
# Beenden des Programms
exit;
```