5 Block 5: Kontrolle des Programmflusses mit *Bedingungen* und *Schleifen*

5.1 Lernziele

Ziel dieses Blocks ist es zu verstehen, wie der Programmfluss in `Perl` mithilfe von *Bedingungen* und *Schleifen* und den assoziierten *Tests* kontrolliert werden können. Im Detail werden folgende programmatischen und syntaktischen Elemente behandelt:

- Die `if...else` Bedingung;
- Die `if...elsif...else` Bedingung;
- Schleifen des Typs `while`;
    - *Mustererkennung* in Schleifen mit *regulären Ausdrücken* `$x =~ /^>/`;
    - Der `next` Befehl zum *Springen ans Schleifenende*;
- Schleifen des Typs `foreach`;
- Schleifen des Typs `do...until`.

5.2 Praxis

Wir haben jetzt alle essentiellen Datentypen in `Perl` kennengelernt: *Skalare*, *Arrays* und *Hashes*. Wir haben ebenfalls einige Methoden kennengelernt, wie diese Daten im Kontext der Sequenzanalyse eingesetzt und manipuliert werden können.

Alle Programme, die wir bisher betrachtet und geschrieben haben, weisen einen mehr oder weniger ununterbrochenen, linearen *Programmfluss* auf. D.h.

- die einzelnen Zeilen des Programmtextes wurden nur einmal ausgeführt;
- und die Ausführung der Programmteile war unabhängig von Bedingungen, die während des Programmablaufs an den Dateninhalt gestellt werden.

Wir werden im folgenden zwei Gruppen von programmatischen Elementen in `Perl` kennenlernen, die eine derartige *Kontrolle des Programmflusses* erlauben:

- *Schleifen* zur wiederholten Ausführung von Teilen eines Programms;
- *Bedingungen* zur bedingten Ausführung alternativer Teile eines Programms.

5.2.1 Bedingungen in `Perl`

Eine *Bedingung* (engl. *conditional statement*, also *konditionaler Ausdruck*) führt einen bestimmten Programmteil nur dann aus, wenn eine definierte Bedingung erfüllt ist. Wenn diese Bedingung nicht erfüllt ist wird der respektive Programmteil übersprungen.

> Also: *Wenn* Bedingung X erfüllt, *dann* führe Programmteil Y aus. *Wenn* Bedingung X nicht erfüllt, *dann* führe Programmteil Y nicht aus – Überspringen!

Die Ausführung ist also immer an eine definierte Formulierung der Bedingung gekoppelt. Diese Bedingung wird auch manchmal als *Test* bezeichnet.

Ein *Test* hat entweder das Ergebnis *wahr* (Bedingung erfüllt) oder *falsch* (Bedingung nicht erfüllt) - engl. und in `Perl` Notation: `true` oder `false`). Wenn der Wert, den der Test zurückgibt `true` ist wird der an die Bedingung geknüpfte Programmteil ausgeführt. Wenn der Wert, den der Test zurückgibt `false` ist, wird er übersprungen.

5.2.1.1 Die `if...else` Bedingung

Betrachten wir folgendes einfache Beispiel `If_else_test.pl` (der Teil mit der `if...else` Bedingung ist gelb hervorgehoben):

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

# Wir initialisieren unseren Hash von Enzymdaten mit einem
# Schluessel-Wert-Paar:
%Enzyme = (
           'Acc16I'   => 'TGCGCA',);

# Der Benutzer wird aufgefordert ein Zeichenkette einzugeben,
# um den Inhalt des Hashes abzufragen:
print "Enzym fuer Abfrage auswaehlen: ";
$Abfrage= <STDIN>;
chomp $Abfrage;

# Wenn der Wert von $Abfrage gleich ("eq" Operator) der Zeichenkette 'Acc16I'
# ist, dann gibt der Test den Wert "true" zurueck und der erste Block wird
# ausgefuehrt.
if ($Abfrage eq 'Acc16I') {
    print "Die Erkennungssequenz fuer $Abfrage ist $Enzyme{Acc16I}\n";
}
# Wenn der Wert von $Abfrage ungleich der Zeichenkette 'Acc16I'
# ist, dann gibt der Test den Wert "false" zurueck und der zweite Block wird
ausgefuehrt.
else {
    print "Keine Daten vorhanden fuer Enzym $Abfrage.\n";
```

```
exit;
}
exit;
```

In der Zeile

```
if ($Abfrage eq 'Acc16I') {...}
```

finden sie die Formulierung der *Bedingung*: Wenn der Wert der Variablen `$Abfrage` gleich der Zeichenkette `Acc16I` ist, dann ist die Bedingung erfüllt (der Test gibt den logischen Wert `true` zurück) und der `if{...}` Block wird ausgeführt, sonst wird der `else{...}` Block ausgeführt.

Beachten sie, dass der Programmblock, der konditional ausgeführt werden soll, immer in geschweiften Klammern `{...}` steht.

Gleichheit von Zeichenketten (Strings) wird mit dem `eq` Operator getestet:

```
if ($Abfrage eq 'Acc16I') {...}
```

Numerische Gleichheit wird mit dem == Operator, *numerische Ungleichheit* mit dem != Operator, eine *numerische Grösser-Als-Beziehung* mit > und eine *numerische Kleiner-Als-Beziehung* mit < ausgedrückt.

```
if ($Zahl == 0) {...} # "true" wenn der Wert von $Zahl gleich 0.
if ($Zahl != 1) {...} # "true" wenn der Wert von $Zahl ungleich 0.
if ($Zahl > 0) {...} # "true" wenn der Wert von $Zahl groeser 0.
if ($Zahl < 0) {...} # "true" wenn der Wert von $Zahl kleiner 0.
```

Analog steht >= für *Grösser-oder-Gleich* und <= für *Kleiner-oder-Gleich*.

Im Programmbeispiel `Enzym_hash_auswahl.pl` haben wir bereits eine `if...else` Bedingung formuliert:

```
%Enzyme = (
        'Acc16I'   => 'TGCGCA',
        'BamHI'    => 'GGATCC',
        'BciVI'    => 'GTATCC',
        'BmrI'     => 'ACTGGG',);
$Auswahl = 'BamHI';
if (exists ($Enzyme{$Auswahl})) {
        print "Die Erkennungssequenz fuer $Auswahl ist $Enzyme{BamHI}\n";
}
else {
        print "Kein Wert vorhanden fuer $Auswahl.";
        exit;
}
```

Mit dem `if (exists ($Enzyme{$Auswahl})` Ausdruck wird getestet ob in dem Hash `%Enzyme` ein Schlüssel mit dem Wert der Variablen `$Auswahl` also `$Enzyme{$Auswahl}` definiert ist. Wenn diese Bedingung erfüllt ist, also der entsprechende Schlüssel vorhanden ist wird der `if{...}` Block sonst der `else{...}` Block ausgeführt. Da in dem Beispiel der Wert von `$Auswahl` der String `BamHI` ist, gibt der Test den Wert `true` zurück, die Bedingung ist also erfüllt und der `if{...}` Block wird ausgeführt.

5.2.1.2 Die `if...elsif...else` Bedingung

Die Bedingung `if...elsif...else` funktioniert analog zu `if...else`. Statt zwei alternativer Blöcke, können mit `if...elsif...else` mehrere Blöcke verwendet und mit Tests versehen werden. Betrachten sie folgenden Programmausschnitt (der `if...elsif...else` ist gelb hervorgehoben):

```
%Enzyme = (
        'Acc16I'   => 'TGCGCA',
        'BamHI'    => 'GGATCC',
        'BciVI'    => 'GTATCC',
        'BmrI'     => 'ACTGGG',);
print "Enzym fuer Abfrage auswaehlen: ";
```

```
$Abfrage= <STDIN>;
if ($Abfrage eq 'Acc16I') {
       print "Die Erkennungssequenz fuer Acc16I ist $Enzyme{Acc16I}\n";
}
elsif ($Abfrage eq 'BamHI') {
       print "Die Erkennungssequenz fuer BamHI ist $Enzyme{BamHI}\n";
}
elsif ($Abfrage eq 'BciVI') {
       print "Die Erkennungssequenz fuer BciVI ist $Enzyme{BciVI}\n";
}
elsif ($Abfrage eq 'BmrI') {
       print "Die Erkennungssequenz fuer BmrI ist $Enzyme{BmrI}\n";
}
else {
       print "Kein Wert vorhanden fuer Enzym $Auswahl.\n";
       exit;
}
```

Für die Formulierung der Bedingung / des Test gilt die gleiche Syntax wie bei `if...else`.

5.2.1.3 Übung

5.2.1.3.1 Aufgabe

- Führen sie das Programm `If_else_test.pl` auf ihrem Computer aus. Beobachten sie die Ausgabe bei verschiedenen Eingaben und studieren sie den Programmtext;
- Schreiben sie auf der Basis von `If_else_test.pl` und `Enzym_hash_auswahl.pl` ein neues Programm `If_elsif_test.pl`, das folgendes Leisten soll:
    - Initialisierung eines Hashes `%Enzyme` mit den vier Schlüssel-Wert-Paaren

`Acc16I`	`TGCGCA`
`BamHI`	`GGATCC`
`BciVI`	`GTATCC`
`BmrI`	`ACTGGG`

    - Einmaliges Abfragen des Vorhandenseins eines *Wertes* (Erkennungssequenz) für einen bestimmten *Schlüssel* (Enzym-Name). Den Enzym-Namen soll der Benutzer über die Kommandozeile eingeben;
    - Formulierung einer `if...elsif..else` Bedingung:
        - Wenn der eingegebene Enzym Name gleich einem der vier Schlüssel von `%Enzyme` dann Ausgabe einer Zeile des Typs:

          Erkennungssequenz von [Enzym Name] ist [Enzym Sequenz].

          (zu schreiben in der Form eines `if{...}...elsif{...}` `...elsif{...}...elsif {...}` Blocks).
        - Wenn der eingegebene Enzym Name ungleich einem der drei Schlüssel von `%Enzyme` dann Ausgabe einer Zeile des Typs:

          Keine Daten fuer Enzym X vorhanden.
          Wollen sie die Daten eingeben (ja / nein)?

          (zu schreiben als `else {...}` Block).

          Innerhalb dieses `else {...}` Blocks soll ein zweiter `if {...}` `else{...}` Block formuliert werden. Wenn der Benutzer mit der Eingabe `ja` antwortet, soll die Zeile

          Erkennungssequenz fuer [Enzym Name] bitte eingeben:

          auf der Kommandozeile ausgeben werden. Daraufhin soll das neue Schlüssel-Wert-Paar in den Hash geschrieben werden. Auf dem Bildschirm soll schliesslich noch eine Ausgabe gemacht werden:

          Erkennungssequenz des neu eingegebenen Enzyms [Enzym Name] ist [Enzym Sequenz].

Wenn der Benutzer mit `nein` antwortet soll das Programm beendet werden.

- Die Bildschirmausgabe soll ungefähr wie folgt gestaltet werden:

```
Erkennungssequenz fuer welches Enzym suchen? HindIII
Keine Erkennungssequenz fuer HindIII definiert.
Wollen sie die Daten eingeben (ja / nein)? ja
Erkennungssequenz fuer HindIII bitte eingeben: ATTCGT
Erkennungsequenz des neuen Enzyms HindIII ist ATTCGT.
```

5.2.1.3.2 Lösung

Programmtext von `If_elsif_test.pl`:

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

# Programm-Name: If_elsif_test.pl
# Autor: picker
# Datum: 11/03
# Aufruf: perl If_elsif_test.pl

# Dieses Programm enthaelt einen Hash mit Restriktionsenzym-Daten und prueft
# das Vorhandensein einer Erkennungssequenz fuer einen vom Benutzer ueber die Komman-
# dozeile eingegebenen Enzym-Namen. Wenn der Name einem Schluessel in dem Hash
# entspricht werden der Name und die Sequenz ausgegeben. Wenn nicht kann der Benutzer
# die Sequenz fuer das neue Enzym eingeben und im Hash speichern.

# Deklaration und Initialisierung des Hashes %Enzyme ('Enzym' => 'Erkennungssequenz'):
%Enzyme = (
            'Acc16I'   => 'TGCGCA',
            'BamHI'    => 'GGATCC',
           'BciVI'    => 'GTATCC',
           'BmrI'     => 'ACTGGG',);
# Pruefen der Erkennungssequenz fuer einen beliebigen Enzym-Namen:
print "Erkennungssequenz fuer welches Enzym suchen? ";
$Such_Enzym = <STDIN>;
chomp $Such_Enzym;
# Wenn der Schluessel vorhanden:
if ($Such_Enzym eq 'Acc16I') {
    print "Erkennungssequenz fuer Acc16I: $Enzyme{Acc16I}\n";
}
elsif ($Such_Enzym eq 'BamHI') {
    print "Erkennungssequenz fuer BamHI: $Enzyme{BamHI}\n";
}
elsif ($Such_Enzym eq 'BciVI') {
    print "Erkennungssequenz fuer BciVI: $Enzyme{BciVI}\n";
}
elsif ($Such_Enzym eq 'BmrI') {
    print "Erkennungssequenz fuer BmrI: $Enzyme{BmrI}\n";
}
# Wenn der Schluessel nicht vorhanden ist:
else {
    print "Keine Erkennungssequenz fuer $Such_Enzym definiert.\n";
    # Auswahl ob Eingabe der Erkennungsseqeunz:
    print "Wollen sie die Daten eingeben (ja / nein)? ";
    $Auswahl = <STDIN>;
    chomp $Auswahl;
    # Wenn Auswahl "ja"
    if ($Auswahl eq 'ja') {
        # Auffoderung die neue Sequenz einzugeben:
        print "Erkennungssequenz fuer $Such_Enzym bitte eingeben: ";
        $Enzym_neu = <STDIN>;
        chomp $Enzym_neu;
        # Zuweisung des neuen Schluessel-Wert-Paares an den Hash:
        $Enzyme{$Such_Enzym} = $Enzym_neu;
        print "Erkennungsequenz des neuen Enzyms $Such_Enzym ist $Enzyme{$Such_Enzym}.\n";
    }
    else {
        print "Programm beendet.\n";
        exit;
    }
}
```

```
exit;
```

5.2.2 Schleifen in `Perl`

Eine *Schleife* (engl. *loop*) wiederholt einen bestimmten Programmteil eingeschlossen in geschweifte Klammern `{...}` solange, bis eine definierte Bedingung *nicht mehr erfüllt* ist (also ein Test den Wert `false` produziert).

> Also: *Solange* Bedingung X erfüllt (der assoziierte Test den Wert `true` produziert), *wiederhole* Programmteil Y. *Wenn* Bedingung X nicht mehr erfüllt ist (der assoziierte Test den Wert `false` produziert) beende den Programmteil – die *Schleife*.

5.2.2.1 Schleifen des Typs `while`

Betrachten wir noch einmal das Programmbeispiel `Seq_In_Dat.pl`:

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

# Dieses Program liest den Inhalt der FASTA Sequenzdatei "Test.fasta" im
# aktuellen Arbeitsverzeichnis ein und gibt ihn auf den Bildschirm aus.

# Definition des Namens der Datei, die geoeffnet werden soll
$Datei_Name = 'Test.fasta';
print "Der Inhalt von $Datei_Name ist:\n\n";
# Offnen der Datei zum Lesen mit dem  "<" Operator
open (FILE, "<$Datei_Name");
# Zuweisung des Dateinhalts an die Array-Variable @Datei_Inhalt
@Datei_Inhalt = <FILE>;
close FILE;
# Ausgabe des Variablen-Werts auf den Bildschrim
print @Datei_Inhalt;
exit;
```

Mit diesem Programm haben wir den Inhalt eine FASTA Datei in einen Array eingelesen und auf dem Bildschirm ausgegeben. Wir mussten den Daten Array `@Datei_Inhalt` verwenden weil die Zuweisung an eine String-Variable `$Datei_Inhalt` nur die erste Zeile gespeichert hat.

Mit einer `while` Schleife können wir diese Problem umgehen und den Datei-Inhalt solange in eine String-Variable einlesen, bis kein zu lesenden Zeilen mehr vorhanden sind. Betrachten sie das modifizierte Programm `Seq_In_Dat_while.pl`:

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

# Definition des Namens der Datei, die geoeffnet werden soll
$Datei_Name = 'Test.fasta';
print "Der Inhalt von $Datei_Name ist:\n\n";
# Offnen der Datei zum Lesen mit dem  "<" Operator
open (FILE, "<$Datei_Name");
# Lesen jeder Zeile in einer Schleife:
while ($Zeile = <FILE>) {
    print $Zeile;
}
close FILE;
exit;
```

Der Test ist in `while ($Zeile = <FILE>)` formuliert. Die Bedingung für die Ausführung der Schleife ist also: Ausführen solange die Zuweisung `$Zeile = <FILE>` einen Wert produziert, in anderen Worten: solange noch Zeilen aus der Datei gelesen werden können. Wenn die Bedingung erfüllt ist (der Wert des Tests `$Zeile = <FILE>` also `true` ist): lese die nächste Zeile (Weiterführen der Schleife). Wenn nicht: *Abbruch* der Schleife.

5.2.2.1.1 Mustererkennung mit regulären Ausdrücken `$x =~ /^>/`

Mit einer einfachen Abänderung können wir erreichen, dass der reine Sequenzinhalt (die *Roh-Sequenz* in gelb) der Datei `Test.fasta`

```
>Test.fasta; eine einfache Testsequenz in FASTA Format
```

```
AATGCTTGCGGGCTATATGCCTTGACGATTTGGGCCCGAT
ACCACCAC
```

eingelesen wird. Betrachten sie das Programmbeispiel `Seq_In_Dat_raw.pl`:

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

# Definition des Namens der Datei, die geoeffnet werden soll
$Datei_Name = 'Test.fasta';
print "Die Roh-Sequenz  von $Datei_Name ist:\n\n";
# Offnen der Datei zum Lesen mit dem  "<" Operator
open (FILE, "<$Datei_Name");
# Lesen jeder Zeile in einer Schleife:
while ($Zeile = <FILE>) {
    if ($Zeile =~ /^>/) {
        next;
    }
    else {
        print $Zeile;
    }
}
close FILE;
exit;
```

In der `while` Schleife von `Seq_In_Dat_raw.pl` finden sie die `if...else` Bedingung mit dem assoziierten Test `($Zeile =~ /^>/)`. Der Test enthält den Binde-Operator =~, der besagt, wende den rechten Teil des Befehls `/^>/` auf den linken Teil `$Zeile` an.

Links steht der String `$Zeile` mit dem momentanen Zeileninhalt. Recht steht der so genannte Musterkennungs-Operator `//`, in diesem Fall als `/^>/`. Der Ausdruck bedeutet also: Wende den Musterkennungs-Operator `/^>/` auf den Wert der Zeichenkette `$Zeile` an. Der Ausdruck `^>` bedeutet: *beginnt-mit-dem-Zeichen* > und wird in `Perl` als *regulärer Ausdruck* bezeichnet (engl. *regular expression*).

Die Testbedingung ist also erfüllt, wenn der Inhalt von `$Zeile` mit dem Zeichen > beginnt. Da in `$Zeile` alle Zeileninhalte gespeichert werden und nur die FASTA Kopfzeile mit > beginnt wird nur deren Ausgabe auf dem Bildschirm in dem `if{...}` Block unterdrückt. Alle anderen Zeilen erfüllen die Bedingung nicht und werden deswegen in dem `else{...}` ausgegeben.

In `Perl` gibt es ein umfangreiches Vokabular regulärer Ausdrücke und eine grossen Satz syntaktischer Regeln wie sie formuliert werden, um Muster in Zeichenketten zu erkennen. Auf die Formulierung und Verwendung regulärer Ausdrücke in `Perl` wird im zweiten Kursteil im Detail eingegangen.

5.2.2.1.2 Der `next` Befehl

Der `next` Befehl bewirkt einen Sprung ans Ende der momentanen Schleife, d.h. wenn die Kopfzeile in `$Zeile` steht (erster Schleifendurchlauf), wird direkt die nächste Zeile im zweiten Durchlauf der Schleife überprüft.

5.2.2.1.3 Übung

5.2.2.1.3.1 Aufgabe

- Führen sie die drei Programme `Seq_In_Dat.pl`, `Seq_In_Dat_while.pl` und `Seq_In_Dat_raw.pl` auf ihrem Computer aus. Beobachten sie die Ausgabe bei verschiedenen Eingaben und studieren sie den Programmtext;
- Schreiben sie auf der Basis von `Seq_In_Dat_raw.pl` ein neues Programm `Enzym_In_Dat_raw.pl`, das analog zu `Seq_In_Dat_raw.pl` nur den eigentlichen Datensatz der Datei `Enzym_Master.tab` einliest und auf dem Bildschirm ausgibt.

5.2.2.1.3.2 Lösung

Programmtext von `Enzym_In_Dat_raw.pl` (gegenüber `Seq_In_Dat_raw.pl` geänderte Teile in gelb):

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

# Definition des Namens der Datei, die geoeffnet werden soll
$Datei_Name = 'Enzym_Master.tab';
print "Der Daten-Inhalt von $Datei_Name ist:\n\n";
# Offnen der Datei zum Lesen mit dem  "<" Operator
open (FILE, "<$Datei_Name");
# Lesen jeder Zeile in einer Schleife:
while ($Zeile = <FILE>) {
    if ($Zeile =~ /^#/) {
        next;
    }
    else {
        print $Zeile;
    }
}
close FILE;
exit;
```

5.2.2.2 Schleifen des Typs `foreach`

Die `foreach` Schleife wird verwendet, um eine Schleife über jeden *skalaren* Einzelwert in einem Array zu bilden.

Wir können `Seq_In_Dat.pl`

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

# Dieses Program liest den Inhalt der FASTA Sequenzdatei "Test.fasta" im
# aktuellen Arbeitsverzeichnis ein und gibt ihn auf den Bildschirm aus.

# Definition des Namens der Datei, die geoeffnet werden soll
$Datei_Name = 'Test.fasta';
print "Der Inhalt von $Datei_Name ist:\n\n";
# Offnen der Datei zum Lesen mit dem  "<" Operator
open (FILE, "<$Datei_Name");
# Zuweisung des Dateinhalts an die Array-Variable @Datei_Inhalt
@Datei_Inhalt = <FILE>;
close FILE;
# Ausgabe des Variablen-Werts auf den Bildschrim
print @Datei_Inhalt;
exit;
```

so modifizieren, dass wir alle Zeilen der eingelesen Datei `Test.fasta` in einer `foreach` Schleife über den Array `@Datei_Inhalt` ausgeben. Betrachten sie `Seq_In_Dat_foreach.pl` (Veränderungen gegenüber `Seq_In_Dat.pl` sind gelb hervorgehoben):

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

# Dieses Program liest den Inhalt der FASTA Sequenzdatei "Test.fasta" im
# aktuellen Arbeitsverzeichnis ein und gibt ihn auf den Bildschirm aus.

# Definition des Namens der Datei, die geoeffnet werden soll
$Datei_Name = 'Test.fasta';
print "Der Inhalt von $Datei_Name ist:\n\n";
# Offnen der Datei zum Lesen mit dem  "<" Operator
open (FILE, "<$Datei_Name");
# Zuweisung des Dateinhalts an die Array-Variable @Datei_Inhalt
@Datei_Inhalt = <FILE>;
close FILE;
# Ausgabe des Variablen-Werts auf den Bildschrim
foreach $Zeile (@Datei_Inhalt) {
    print "Zeile:\t$Zeile";
}
exit;
```

Die `foreach` Schleife kann auch verwendet werden, um eine Schleife über jedes *Schlüssel-Wert-Paar* in einem Hash zu bilden:

```
foreach $Schluessel (keys %Hash) {
    print "$Schluessel\t$Hash{$Schluessel}\n";
```

```
}
```

In diesem Beispiel werden die *Schlüssel-Wert-Paare* tabulator-separiert auf dem Bildschirm ausgegeben.

Der `keys` Befehl (gelb in oben stehendem Beispiel) konstruiert eine Liste aller Schlüssel (engl. *keys*) des Hashes bevor die Schleife startet. Dann wird die Schleife über diese Liste der Schlüssel gebildet. Gleichzeitig wird der Variablen `$Schluessel` immer der nächste Schlüssel mit `keys` zugewiesen, so dass über den Inhalt von `$Schluessel` in der Schleife auf alle Werte mit `$Hash{$Schluessel}` zugegriffen werden kann.

5.2.2.2.1 Übung

5.2.2.2.1.1 Aufgabe

- Führen sie das Programm `Seq_In_Dat_foreach.pl` auf ihrem Computer aus. Beobachten sie die Ausgabe bei verschiedenen Eingaben und studieren sie den Programmtext;
- Schreiben sie ein neues Programm `Ausgabe_Enzym_Hash.pl`, das folgendes Leisten soll:
    - Initialisierung eines Hashes `%Enzyme` mit den vier Schlüssel-Wert-Paaren

```
Acc16I      TGCGCA
BamHI       GGATCC
BciVI       GTATCC
BmrI        ACTGGG
```

- Eine `foreach` Schleife, die jedes Schlüssel-Wert-Paar auf dem Bildschirm ausgibt;
- Die Ausgabe soll ungefähr folgendermassen aussehen:

```
Die aktuellen Daten fuer Restriktionsenzyme:
--------------------------------------------

[Name]  [Erkennungssequenz]

BciVI   GTATCC
BamHI   GGATCC
Acc16I  TGCGCA
BmrI    ACTGGG
```

5.2.2.2.1.2 Lösung

Programmtext von `Ausgabe_Enzym_Hash.pl`:

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

# Deklaration und Initialisierung des Hashes %Enzyme ('Enzym' =>
'Erkennungssequenz'):
%Enzyme = (
            'Acc16I'   => 'TGCGCA',
            'BamHI'    => 'GGATCC',
           'BciVI'    => 'GTATCC',
           'BmrI'     => 'ACTGGG',);
print "\nDie aktuellen Daten fuer Restriktionsenzyme:\n";
print "--------------------------------------------\n\n";
print "[Name]\t[Erkennungssequenz]\n\n";
# foreach Schleife ueber den Hash %Enzyme zur Ausgabe auf dem Bildschirm:
foreach $Schluessel (keys %Enzyme) {
    print "$Schluessel\t$Enzyme{$Schluessel}\n";
}
```

5.2.2.3 Schleifen des Typs `do...until`

Der letzte `Perl`-Schleifentyp, den wir besprechen ist die `do...until` Schleife. In einer solchen Schleife wird ein Programmblock ausgeführt und anschliessend eine Bedingung gestestet. Wenn die Bedingung erfüllt ist, wird der Block nochmals ausgeführt usw.

Betrachten wir das Programmbeispiel `Do_Until_test.pl` (die `do...until` Schleife ist gelb hervorgehoben, die `foreach` Schleife in blau):

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

# Programm-Name: Do_Until_test.pl
# Autor: picker
# Datum: 11/03
# Aufruf: perl Do_Until_test.pl

# Aufforderung eine neue Oligonukleotid-Sequenz einzugeben:
print "Sequenz fuer das Oligonukleotid eingeben: ";
$Oligo_Sequenz = <STDIN>;
chomp $Oligo_Sequenz;
# Aufforderung einen Namen fuer die Oligonukleotid-Sequenz einzugeben:
print "Name fuer das Oligonukleotid eingeben: ";
$Oligo_Name = <STDIN>;
chomp $Oligo_Name;
# Anhaengen der Namens und der eingegebenen Sequenz einen Array zu
Speicherung:
push (@Alle_Oligos, $Oligo_Name);
push (@Alle_Oligos, $Oligo_Sequenz);
# Ausgabe der Daten auf dem Bildschirm:
print "Die Sequenz von $Oligo_Name ist: $Oligo_Sequenz\n";
# Auffoderung auszuwaehlen ob noch ein Datensatz eingegeben werden soll:
print "Ein weiteres Oligonukleotid eingeben (ja / nein)? ";
$Auswahl = <STDIN>;
# Ausfuehren des do{} Blocks:
do {
    print "Sequenz fuer das Oligonukleotid eingeben: ";
    $Oligo_Sequenz = <STDIN>;
    chomp $Oligo_Sequenz;
    print "Name fuer das Oligonukleotid eingeben: ";
    $Oligo_Name = <STDIN>;
    chomp $Oligo_Name;
    push (@Alle_Oligos, $Oligo_Name);
    push (@Alle_Oligos, $Oligo_Sequenz);
    print "Die Sequenz von $Oligo_Name ist: $Oligo_Sequenz\n";
    print "Ein weiteres Oligonukleotid eingeben (ja / nein)? ";
    $Auswahl = <STDIN>;
} until ($Auswahl =~ /nein/);
# Bis die Eingabe in $Auswahl den String 'nein' enthaelt.
print"\nDie Liste der der eingebenen Oligoukleotide:\n";
print"-----------------------------------------\n\n";
foreach $Array_Element (@Alle_Oligos) {
    print "$Array_Element\n";
}
exit;
```

Dieses Programm fordert den Benutzer auf

- Schritt (1): eine Sequenz und einen Namen einzugeben;
- Schritt (2): gibt beides auf der Kommandozeile aus;
- Speichert beides in einem Array;
- Wiederholt Schritt (1) und (2) in einer `do...until` Schleife solange bis der Benutzer auf die Aufforderung "`Ein weiteres Oligonukleotid eingeben (ja / nein)?`" mit `nein` antwortet;
- Und gibt anschliessend alle eingegebenen Daten in einer `foreach` Schleife auf dem Bildschirm aus.

In einer `do...until` Schleife wird also im Gegensatz zu den andern Schleifen und Bedingungen der Test erst nach dem ersten Schleifendurchlauf durchgeführt.

In unserem Beispiel ist der Test durch `$Auswahl =~ /nein/` nach dem `until` Ausdruck formuliert: die Schleife bricht ab wenn die Variable `$Auswahl` eine Wert hat, der das Muster `nein` enthält.

5.2.2.3.1 Übung

5.2.2.3.1.1 Aufgabe

- Führen sie das Programm `Do_Until_test.pl` auf ihrem Computer aus. Beobachten sie die Ausgabe bei verschiedenen Eingaben und studieren sie den Programmtext;
- Schreiben sie auf der Basis von `Do_Until_test.pl` und `If_elsif_test.pl` ein neues Programm `Addieren_Enzym_Hash.pl`, das folgendes Leisten soll:
  - Deklaration eines leeren Hashes `%Enzyme()`.
  - Eine `do...until` Schleife, in der beliebig viele Schlüssel-Wert-Paare des Typs

    `Enzym-Name => Erkennungssequenz`

    über die Befehlszeile eingeben werden können und in `%Enzyme` gespeichert werden;
  - Ausgabe aller Schlüssel-Wert-Paare analog zu der Ausgabe von `Ausgabe_Enzym_Hash.pl` (schreiben sie dazu eine `foreach` Schleife über `%Enzyme` wie in `Ausgabe_Enzym_Hash.pl`):

```
Die aktuellen Daten fuer Restriktionsenzyme:
--------------------------------------------

[Name]  [Erkennungssequenz]

BciVI   GTATCC
BamHI   GGATCC
Acc16I  TGCGCA
BmrI    ACTGGG
```

5.2.2.3.1.2 Lösung

Programmtext von `Addieren_Enzym_Hash.pl` (`do...until` Schleife in gelb, `foreach` Schleife in blau):

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

# Programm-Name: Addieren_Enzym_Hash.pl
# Autor: picker
# Datum: 11/03
# Aufruf: perl Addieren_Enzym_Hash.pl

# Deklaration eines leeren Hashes:
%Enzyme = ();

# do...until Schleife fuer die wiederholte Dateneingabe:
do {
    # Eingabe des Namens (Schluessel):
    print "Name fuer das Enzym eingeben: ";
    $Enzym_Name = <STDIN>;
    chomp $Enzym_Name;
    # Eingabe der Sequenz (Wert):
    print "Erkennungssequenz fuer das Enzym $Enzym_Name eingeben: ";
    $Enzym_Sequenz = <STDIN>;
    chomp $Enzym_Sequenz;
    # Zuweisung des Schluessel(Name)-Wert(Sequenz)-Paares an den Hash:
    $Enzyme{$Enzym_Name} = $Enzym_Sequenz;
    print "Die Erkennungssequenz fuer $Enzym_Name ist:\t$Enzyme{$Enzym_Name}\n";
    # Aufforderung zu bestaetigen, dass ein neues Enzym eingegeben werden soll:
    print "Daten fuer ein weiteres Enzym eingeben (ja / nein)? ";
    $Auswahl = <STDIN>;
}until ($Auswahl =~ /nein/);
# Bis die Eingabe in $Auswahl den String 'nein' enthaelt.
print "\nDie aktuellen Daten fuer Restriktionsenzyme:\n";
print "--------------------------------------------\n\n";
print "[Name]\t[Erkennungssequenz]\n\n";
# foreach Schleife ueber den Hash %Enzyme zur Ausgabe auf dem Bildschirm:
foreach $Schluessel (keys %Enzyme) {
    print "$Schluessel\t$Enzyme{$Schluessel}\n";
}
```