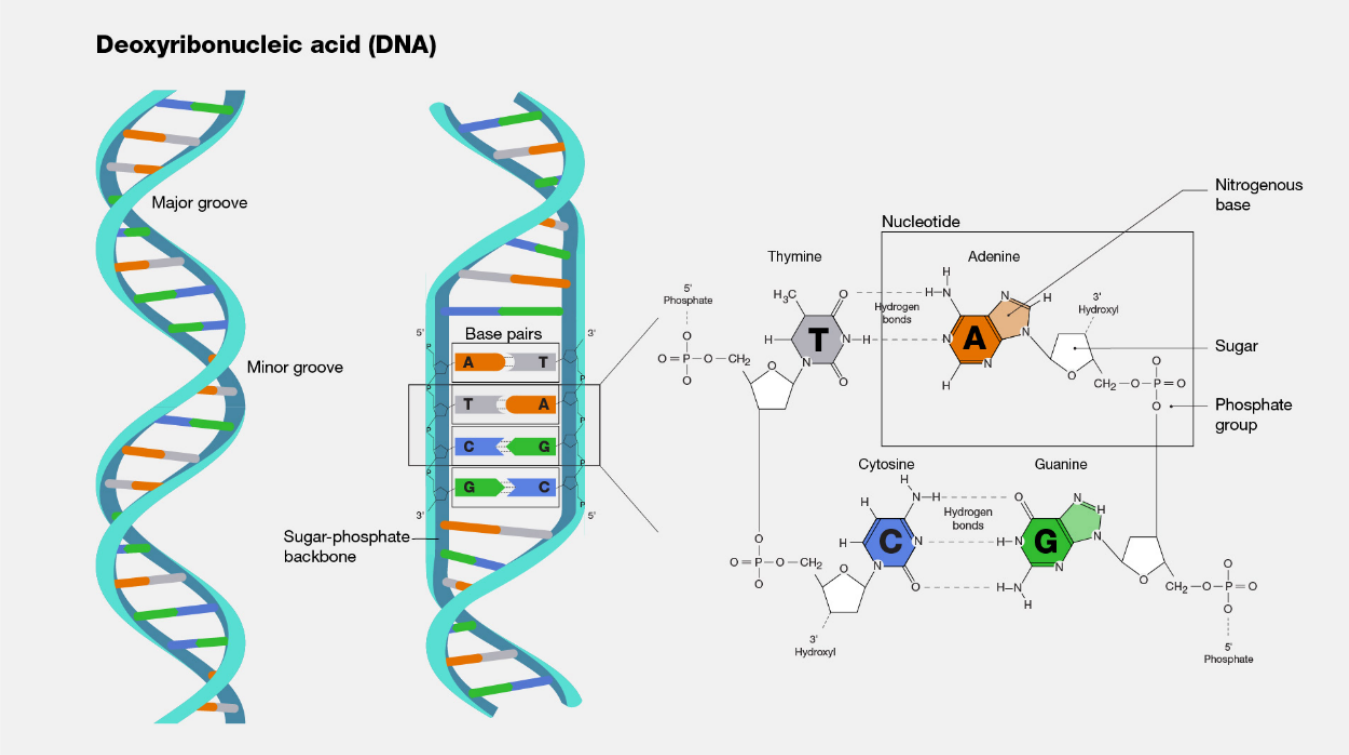
Biopython

Biopython ist ein Paket mit einer vielzahl von Werkzeugen zur Berechnung biologischer Daten.

Erinnerung: Aufbau von DNA

Die DNA ist als Doppelstrang aufgebaut und gewunden, ähnlich einer gewundenen Leiter. Die Sprossen der Leiter entsprechen den Basenpaaren, welche wiederum durch die Verbindung zweier Nukleotide gebildet werden. Adenin mit Thymin und Guanin mit Cytosin bilden hierbei jeweils die Paare. Eine Sequenz bezeichnet hierbei eine Abfolge von Nukleotiden entlang eines Strangs der DNA. Den gegenüberliegenden Strang, mit dem die Paare gebildet werden, nennt man komplementär.



Arbeiten mit Sequenzen

In [1]:

```
import ipywidgets as wg
from IPython.display import display
```

In [2]:

```
from Bio.Seq import Seq
from Bio import SeqIO
import matplotlib.pylab as pylab
```

In [3]:

```
my_seq = Seq("AGTACACTGGT") # Sequenz festlegen
my_seq                      # Sequenz anzeigen
```

Out[3]:

```
Seq('AGTACACTGGT')
```

In [4]:

```
my_seq.complement() # komplementäre Sequenz anzeigen
```

Out[4]:

```
Seq('TCATGTGACCA')
```

In [5]:

```
my_seq.reverse_complement() # komplementäre Sequenz in umgekehrter Reihenfolge anzeigen
```

Out[5]:

```
Seq('ACCAGTGTACT')
```

Parsing

Beim Parsing wird ein Objekt in seine Einzelteile zerlegt. So können wir in unserem Fall Bestandteile wie ID, die Sequenz selbst oder die Länge einer Sequenz einzeln abfragen.

FASTA-Format

In [6]:

```
for seq_record in SeqIO.parse('data/ls_orchid.fasta.txt', 'fasta'):
    print(seq_record.id) # ID anzeigen
    print(repr(seq_record.seq)) # Sequenz anzeigen
    print(len(seq_record)) # Länge der Sequenz anzeigen
```

```
726
gi|2765645|emb|Z78520.1|CSZ78520
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGACAGCAG...TTT')
753
gi|2765644|emb|Z78519.1|CPZ78519
Seq('ATATGATCGAGTGAATCTGGTGGACTTGTGGTTACTCAGCTCGCCATAGGCTTT...TTA')
699
gi|2765643|emb|Z78518.1|CRZ78518
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGGAGGATCATTGTTGAGATAGTAG...TCC')
658
gi|2765642|emb|Z78517.1|CFZ78517
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGACAGTAG...AGC')
752
gi|2765641|emb|Z78516.1|CPZ78516
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGACAGTAT...TAA')
726
gi|2765640|emb|Z78515.1|MXZ78515
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGCTGAGACCGTAG...AGC')
765
gi|2765639|emb|Z78514.1|PSZ78514
```

GenBank-Format

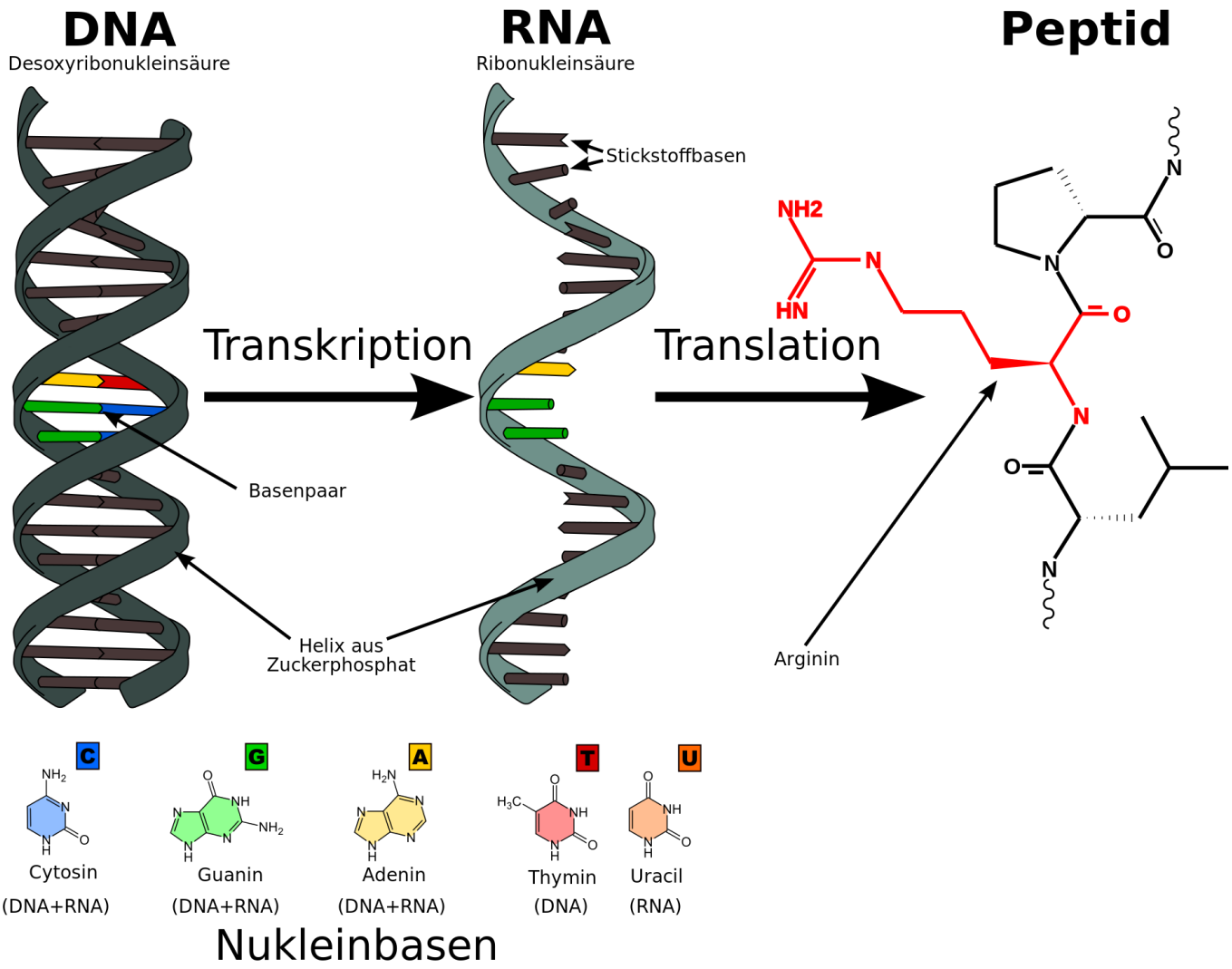
In [7]:

```
for seq_record in SeqIO.parse('data/ls_orchid.gbk.txt', 'genbank'):
    print(seq_record.id) # ID anzeigen
    print(repr(seq_record.seq)) # Sequenz anzeigen
    print(len(seq_record)) # Länge der Sequenz anzeigen
```

```
Z78533.1
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGATGAGACCGTGG...CGC')
740
Z78532.1
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGACAACAG...GGC')
753
Z78531.1
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGACAGCAG...TAA')
748
Z78530.1
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAAACAACAT...CAT')
744
Z78529.1
Seq('ACGGCGAGCTGCCGAAGGACATTGTTGAGACAGCAGAATATACGATTGAGTGAA...AAA')
733
Z78527.1
Seq('CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGACAGTAG...CCC')
718
Z78526.1
```

Proteinbiosynthese

Ein grober Einblick in den Vorgang der Proteinbiosynthese um die beiden folgenden Punkte besser nachvollziehen zu können.



Die erste Phase bei der Proteinbiosynthese ist die Transkription. Hierbei wird DNA in mRNA umgewandelt. Der Doppelstrang der DNA wird dazu aufgespalten und der sogenannte codogene Strang dient als Vorlage für die Übersetzung in die mRNA. In der mRNA wird die Nukleinbase Thymin allerdings zu Uracil.

In der zweiten Phase, der Translation, werden in den Ribosomen mithilfe der mRNA Proteine hergestellt.

Sequenzen verhalten sich wie Strings

In [8]:

```
# Iteration der Sequenz
my_seq = Seq('GATCG')
for index, letter in enumerate(my_seq):
    print('%i %s' % (index, letter))
```

```
0 G
1 A
2 T
3 C
4 G
```

In [9]:

```
print(len(my_seq)) # Länge der Sequenz
```

```
5
```

In [10]:

```
Seq('AABBAACC').count('AA') # Zählung wie oft AA vorkommt
```

Out[10]:

```
2
```

GC-Gehalt berechnen

Der GC-Gehalt gibt an wie viele Guanin und Cytosin-Moleküle in einer Nukleinsäure vorkommen und dient der Bestimmung der Bindungsfähigkeit und des Energiegehalts.

In [11]:

```
from Bio.SeqUtils import gc_fraction
my_seq = Seq("GATCGATGGGCCTATATAGGATCGAAAATCGC")
gc_fraction(my_seq)
```

Out[11]:

```
0.46875
```

Sequenzen zerschneiden

In [12]:

```
my_seq = Seq("GATCGATGGGCCTATATAGGATCGAAAATCGC")
my_seq[4:12]                                    # Sequenz von der 5. bis zur 12. Stel
```

Out[12]:

```
Seq('GATGGGCC')
```

In [13]:

```
my_seq[::3]  # Gibt jeden dritten Buchstaben aus, Start an stelle 0 ( [Start:Stop:Steps]
```

Out[13]:

```
Seq('GCTGTAGTAAG')
```

Zusammenfügen von Sequenzen

In [14]:

```
seq1 = Seq("ACGT")
seq2 = Seq("AACCGG")
seq1 + seq2
```

Out[14]:

```
Seq('ACGTAACCGG')
```

Verkettung mit Hilfe einer For-Schleife

In [15]:

```
list_of_seqs = [Seq("ACGT"), Seq("AACC"), Seq("GGTT")] # Liste von Sequenzen definieren
verkettung = Seq("")
for s in list_of_seqs:  # For-Schleife zur Verkettung der Einzelsequenzen
    verkettung += s
```

In [16]:

```
verkettung
```

Out[16]:

```
Seq('ACGTAACCGGTT')
```

.join Methode

Die .join Methode nimmt alle Teile eines iterierbaren Objekts und fügt sie zu einem String zusammen. Außerdem ist es mit dieser Methode möglich, einen Separator zwischen den Einzelsequenzen zu ergänzen.

In [17]:

```
contigs = [Seq("ATG"), Seq("ATCCCG"), Seq("TTGCA")] # Liste von Sequenzen definieren
spacer = Seq("N" * 10) # Separator definieren
spacer.join(contigs)
```

Out[17]:

```
Seq('ATGNNNNNNNNNNATCCCGNNNNNNNNNNTTGCA')
```

Transkrtiption

In [18]:

```
coding_dna = Seq("ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG") # Definition der Sequenz
coding_dna
```

Out[18]:

```
Seq('ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG')
```

In [19]:

```
messenger_rna = coding_dna.transcribe() # Transkription der Sequenz
messenger_rna
```

Out[19]:

```
Seq('AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG')
```

Der Buchstabe T wird mit dem Buchstaben U ausgetauscht, Thymin wird also zu Uracil.

In [20]:

```
# Die Transkription kann auch zurückgeführt werden, um von mRNA zur DNA zu gelangen
messenger_rna.back_transcribe()
```

Out[20]:

```
Seq('ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG')
```

Translation

In [21]:

```
messenger_rna = Seq("AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG") # Translation der Sequenz
messenger_rna
```

Out[21]:

```
Seq('AUGGCCAUUGUAAUGGGCCGCUGAAAGGGUGCCCGAUAG')
```

In [22]:

```
messenger_rna.translate() # Übersetzen der mRNA in die dazugehörige Proteinsequenz
```

Out[22]:

```
Seq('MAIVMGR*KGAR*')
```

In [23]:

```
coding_dna
```

Out[23]:

```
Seq('ATGGCCATTGTAATGGGCCGCTGAAAGGGTGCCCGATAG')
```

Es ist auch möglich, direkt vom codogenen DNA-Strang ausgehend zu übersetzen.

In [24]:

```
coding_dna.translate()
```

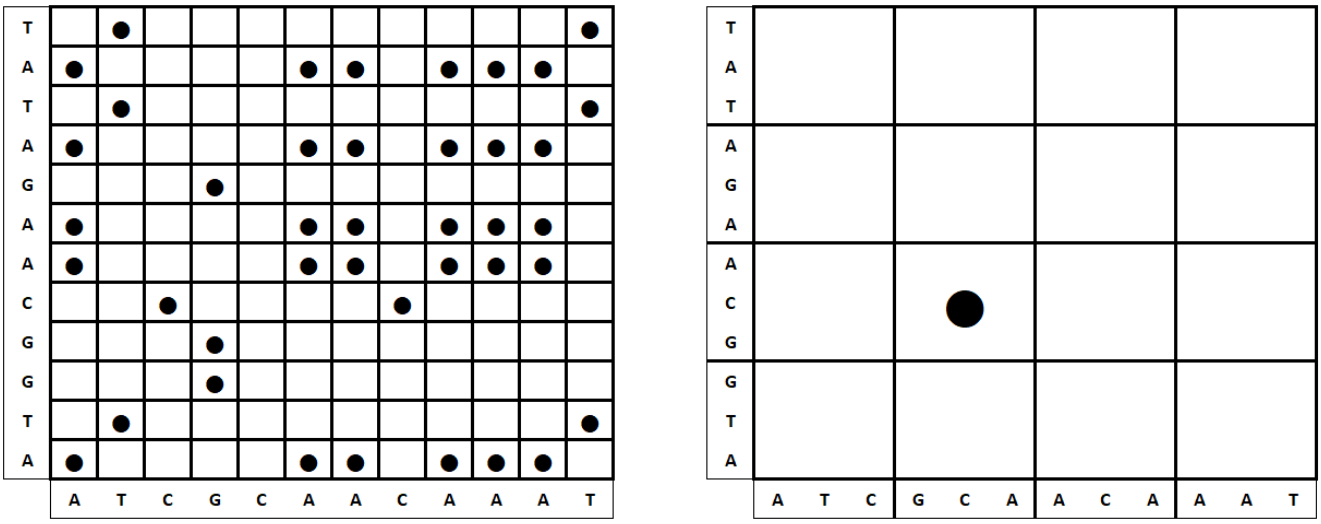
Out[24]:

```
Seq('MAIVMGR*KGAR*')
```

Dotplot

Mit einem Dotplot lassen sich zwei Nukleotidsequenzen visuell auf Übereinstimmungen überprüfen.

Die Sequenzen werden dabei in einem Koordinatensystem gegenübergestellt und bei Übereinstimmungen wird ein Punkt erstellt, wie im folgenden Bild veraunschaulicht. Im Gegensatz zum linken Bild, werden im rechten Bild nicht die einzelnen Nukleotide miteinander verglichen, sondern Sequenzen aus drei Nukleotiden.



Um die Angelegenheit einfach zu halten, wird in diesen Beispielen nur nach exakten Übereinstimmungen gesucht.

Einfache, naive Implementierung

Dieser Ansatz vergleicht alle Teilsequenzen mit einer bestimmten Fenstergröße miteinander. Fenstergröße steht in diesem Fall für die Länge der Teilsequenzen. Es handelt sich hierbei um einen Brute-Force-Ansatz, der Alles mit Allem vergleicht und langsam ist.

In [25]:

```
from Bio import SeqIO

# Die ersten beiden Sequenzen der FASTA- Datei als rec_one und rec_two definieren
with open('data/ls_orchid.fasta.txt') as in_handle:
    record_iterator = SeqIO.parse(in_handle, 'fasta')
    rec_one = next(record_iterator)
    rec_two = next(record_iterator)
```

In [26]:

```
print('Größe des Fensters:')
win = wg.IntSlider( value=7,min=1, max=50)
display(win)
```

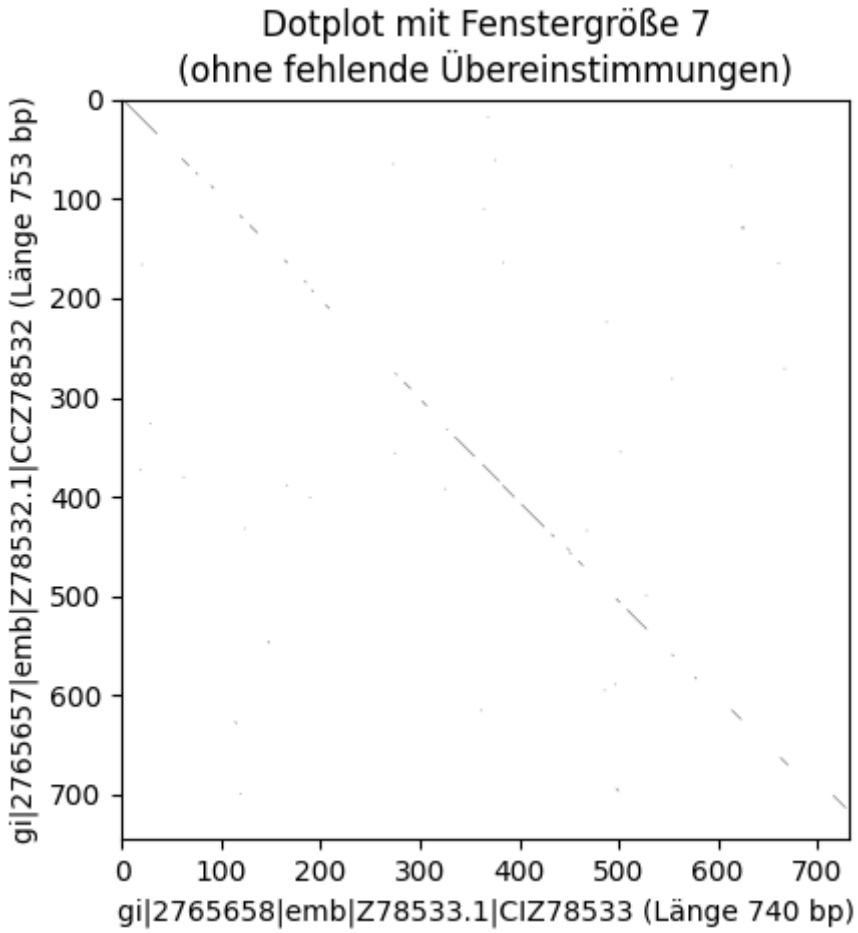
```
Größe des Fensters:

IntSlider(value=7, max=50, min=1)
```

In [27]:

```
window = win.value # Länge der zu vergleichenden Sequenz
# Beide Sequenzen komplett in Großbuchstaben
seq_one = rec_one.seq.upper()
seq_two = rec_two.seq.upper()
# Erstellen einer Liste Boolescher Werte
data = [
    [
        (seq_one[i : i + window] !=seq_two[j : j + window])
        for j in range(len(seq_one) - window)
    ]
    for i in range(len(seq_two) - window)
]
```

In [28]:

```
pylab.gray()
pylab.imshow(data)
pylab.xlabel("%s (Länge %i bp)" % (rec_one.id, len(rec_one)))
pylab.ylabel("%s (Länge %i bp)" % (rec_two.id, len(rec_two)))
pylab.title("Dotplot mit Fenstergröße %i\n(ohne fehlende Übereinstimmungen)" % window)
pylab.show()
```



Fortgeschrittener Ansatz

In diesem Ansatz werden zunächst Dictionaries erstellt, in denen jeder Teilsequenz einer Position zugeordnet ist. Anschließend werden die Überschneidungen der beiden Sets genutzt, um Teilsequenzen zu finden, die in beiden Sequenzen vorkommen. Dieser Ansatz benötigt zwar mehr Speicher, ist jedoch viel schneller.

In [29]:

```
# Erstellen der Dictionaries für beide Sequenzen
window = win.value
dict_one = {}
dict_two = {}
for (seq, section_dict) in [
    (rec_one.seq.upper(), dict_one),
    (rec_two.seq.upper(), dict_two),
]:
    for i in range(len(seq) - window):
        section = seq[i : i + window]
        try:
            section_dict[section].append(i)
        except KeyError:
            section_dict[section] = [i]
```

In [30]:

```
# Finden von Teilsequenzen, die in beiden Sequenzen vorkommen
matches = set(dict_one).intersection(dict_two)
print("%i eindeutige Übereinstimmungen" % len(matches))
```

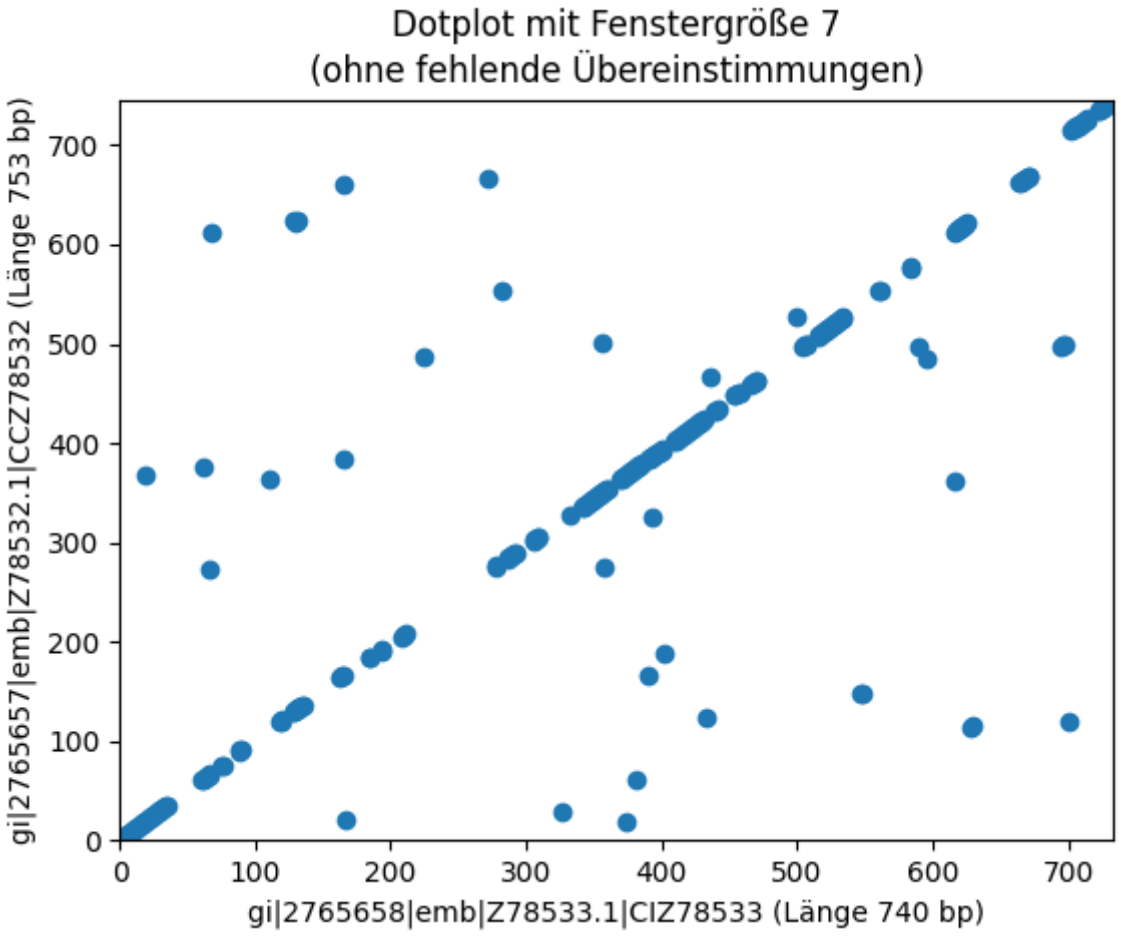
```
244 eindeutige Übereinstimmungen
```

In [31]:

```
# Erstellen einer Liste mit x und y Koordinaten für einen scatter plot
x = []
y = []
for section in matches:
    for i in dict_one[section]:
        for j in dict_two[section]:
            x.append(i)
            y.append(j)
```

In [32]:

```
pylab.cla() # vorherige Graphen Leeren
pylab.gray()
pylab.scatter(x, y)
pylab.xlim(0, len(rec_one) - window)
pylab.ylim(0, len(rec_two) - window)
pylab.xlabel("%s (Länge %i bp)" % (rec_one.id, len(rec_one)))
pylab.ylabel("%s (Länge %i bp)" % (rec_two.id, len(rec_two)))
pylab.title("Dotplot mit Fenstergröße %i\n(ohne fehlende Übereinstimmungen)" % window)
pylab.show()
```



Quellen:
https://de.serlo.org/biologie/70800/proteinbiosynthese-proteine-herstellen
http://biopython.org/DIST/docs/tutorial/Tutorial.pdf
https://facellitate.com/de/deoxyribonucleic-acid-dna-building-block-of-life/
https://www.orchid.inf.tu-dresden.de/research/parsing.de/#:~:text=Der%20Begriff%20Parsing%20beschreibt%20das,zur%20Zerlegung%20von%20Nutzereingaben%20erzeugen
Arnemann, J. (2019). GC-Gehalt. In: Gressner, A.M., Arndt, T. (eds) Lexikon der Medizinischen Laboratoriumsdiagnostik. Springer Reference Medizin. Springer, Berlin, Heidelberg.