

# Assignment solutions

## Bioinformatics for Life Scientists

Die Übungen werden online im ILIAS bearbeitet und müssen jede Woche bis Freitag 23:59 Uhr abgegeben werden. Es gibt das Angebot die Aufgaben im PC-Pool zu bearbeiten. Das lohnt sich vor allem dann, wenn man Fragen an den Tutor hat und nicht weiter weiß oder wenn man die zur Bearbeitung benötigten Programme nicht installieren kann. Gewichtung: 60% Klausur, 40% Übungen (beides muss mit mindestens 50% bestanden werden). Hier die Lösungen aus dem Sommersemester 2017. Manchmal werden auch andere Lösungen akzeptiert, die nicht in der Musterlösung stehen. Außerdem gibt es teils mehrere Varianten einer Teilaufgabe, sodass nicht jeder Student dieselbe Aufgabe angezeigt bekommt.

### Assignment 01: UNIX I

1. **UNIX commands** (1 Punkt)

Assign the following Unix commands to their task:

mkdir → create new directory

rmdir → delete empty directory

cp → copy file or directory

mv → move or rename file or directory

2. **Show file details** (1 Punkt)

How can you show the details (e.g. file size) of an existing file **file.txt**? Please note that Unix commands and file names are **case-sensitive**, do not use more spaces than necessary:

`ls -l file.txt`

3. **Rename a file** (2 Punkte)

How do you rename the file test.txt to test2.txt?

☒ `mv test.txt test2.txt`

☐ `cp test.txt test2.txt`

☐ `ren test.txt test2.txt`

☐ `cat test.txt > test2.txt`

4. **Relative and absolute paths** [Variante 3] (2 Punkte)

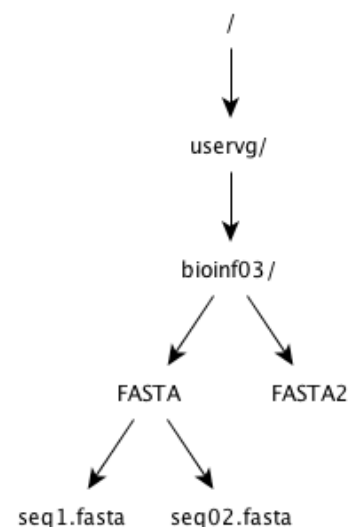
Assume that your home directory is `/uservg/bioinf03` and your current directory is `/uservg/bioinf03/FASTA2`. How can you move `seq1.fasta` and `seq02.fasta` to your current directory?

☒ `mv ../FASTA/*.fasta .`

☐ `mv ../FASTA/*.fasta ..`

☐ `mv ../FASTA/*.fasta ../FASTA2`

☐ `mv ../FASTA/*.fasta FASTA2`



5. **Wildcards** (3 Punkte)

You have three files file01.txt, file02.txt and file10.txt. How can you delete file01.txt and file02.txt, but not file10.txt with a single command?

- ☐ `rm file?.txt`
- ☐ `rm file*.txt`
- ☐ `rm file*?.txt`
- ☒ None of the available choices is correct

6. **Relative and absolute paths** (3 Punkte)

Assume that your home directory is `/uservg/bioinf03` and your current working directory is `/uservg/bioinf03/FASTA`. You want to move both `.fasta` files to your home directory. If more than one answer is correct you must select **all** correct solutions.

- ☐ `mv *.fasta .`
- ☒ `mv *.fasta ..`
- ☒ `mv *.fasta ~`
- ☒ `mv *.fasta /uservg/bioinf03`
- ☐ `mv *.fasta /`
- ☐ `mv seq?.fasta ..`

## Assignment 02: UNIX II / NGS

1. **Group Permissions** (1 Punkt)

You have the following file in your filesystem:

```
-rwxr-xr-- 1 uservg03 BILS 0 Apr 20 12:33 myFile.txt
```

What permissions does the file's **group** have? If more than one answer is correct you must select **all** correct solutions.

- ☒ Read
- ☐ Write
- ☒ Execute
- ☐ None of the other choices

2. **List the contents of an archive** (1 Punkt)

How do you list the contents of an archive `bachelor_thesis.tar.gz` using **tar**? Complete the following command (do not add additional spaces):

```
tar tfz bachelor_thesis.tar.gz
oder tar tzf bachelor_thesis.tar.gz
oder tar ftz bachelor_thesis.tar.gz
oder tar fzt bachelor_thesis.tar.gz
oder tar zft bachelor_thesis.tar.gz
oder tar ztf bachelor_thesis.tar.gz
oder tar tf bachelor_thesis.tar.gz
oder tar ft bachelor_thesis.tar.gz
```

### 3. Working with permissions (2 Punkte)

Create an empty file named **permission\_test.txt** in your home directory using the *touch* command. How can you check which permissions are set on this newly created file?

```
ls -l permission_test.txt
oder ls -la permission_test.txt
oder ls -al permission_test.txt
```

Depending on the settings of your system the permissions on newly created files can differ. You want to enable a fellow student to modify the file you created. Her user account is in the same group as yours. Type the command (with options) that you would use to add write permissions for the **group** of the file **permission\_test.txt**. Do not add any additional spaces before or after your entry.

```
chmod g+w permission_test.txt
```

### 4. grep and wc (2 Punkte)

Download the file **der\_process** (448.64 KB). Use **grep** to find out how many lines contain one or more mentions of the word **Richter**. First use the **man page** of **grep** to find the correct parameter to print the number of matching lines. Enter the complete **grep** command. The output of your solution should be just the number 66:

```
grep -c "Richter" der_process
oder grep "Richter" -c der_process
oder grep "Richter" der_process -c
oder grep -c Richter der_process
oder grep Richter -c der_process
oder grep Richter der_process -c
```

Now use **grep** to print all matching lines and forward them to **wc** using the **pipe operator** **|**. You can then use **wc** to count the number of lines given by **grep**. You have to use a parameter of **wc** to obtain just the number of lines. Complete the following command. Again the output should be just the number 66:

```
grep "Richter" der_process | wc -l
oder grep Richter der_process | wc -l
```

### 5. Variant calling [Variante 2] (6 Punkte)

Note: We recommended to do this task in the computer pool since all required tools are pre-installed. You can find them here, if you prefer to use your own computer (not recommended):

- bowtie2 (<http://bowtie-bio.sourceforge.net/bowtie2/index.shtml>)
- samtools (<http://samtools.sourceforge.net/>)
- tablet (<https://ics.hutton.ac.uk/tablet/download-tablet/>)

Create a new directory and download the files provided above to this directory and extract the archive. This step is recommended to make sure that you work with your own files and not files that another student left on the computer.

In this task we will establish a basic variant calling pipeline to identify SNPs in the reads provided with this task. You will analyze the data of larger NGS studies in a similar way. Our pipeline is based on the one described here: [http://biobits.org/samtools\\_primer.html](http://biobits.org/samtools_primer.html). You can use this link and other web resources to learn more about the commonly used tools bowtie and samtools.

All files necessary for this task are bundled in the archive linked above. These are: A reference genome and the reads from the sequencer.

**Reference genome:** In this task we will use the relatively small genome from *Helicobacter pylori* which is provided as NC\_000915.fna in the archive above. First we want to get an estimation of the genome size.

**Question 1:** Enter a Unix command that will print the genome size (i.e. number of base pairs) of *H. pylori* just by looking at the file NC\_000915.fna provided in this assignment. Your command only needs to give a rough estimation of the genome size (+/- 50 bp). Enter the full Unix command:

```
wc -c NC_000915.fna
oder wc -m NC_000915.fna
```

The **reads** from the sequencer: Reads are usually provided in the FASTQ format which contains a quality score for every base (it is set to 2 for every base in this case). Open the file reads2.fq and [https://en.wikipedia.org/wiki/FASTQ\\_format](https://en.wikipedia.org/wiki/FASTQ_format) to familiarize yourself with the format. How can you count the number of reads in this file? Hint: Each read is preceded by a line containing an identifier. Therefore you could use grep to count the number of identifier lines. The grep parameter -c will print the number of lines that grep matched (see man grep).

**Question 2:** Enter a grep command that will count the exact number of reads in reads2.fq:

```
grep -c @ reads2.fq
oder grep -c @gi reads2.fq
```

We will use the popular tools *bowtie2* and *samtools* to map the reads to the reference genome. To make them accessible in the computer pool open a Terminal and enter the command:

```
source /bils/bashrc
```

If you close the Terminal you will need to enter the command again.

In order to map millions of reads at sufficient speed *bowtie2* needs an index for the reference genome. There are pre-built indexes for some genomes, otherwise you would have to create one yourself. For this assignment we provided an index named *H\_pylori\_index* in the zip archive. Make sure that all index files are in the same folder.

```
bowtie2 -x H_pylori_index -U reads2.fq -S mapped_reads.sam
```

Parameters:

- x: location of the reference genome index files
- U: unpaired reads
- S: mapped reads in SAM (Sequence Alignment/Map) format.

The SAM file that *bowtie2* created is text-based which makes it human-readable but results in large file sizes. Most algorithms will only use the compressed binary BAM format (Binary Alignment/Map). To **convert** our SAM file to a BAM file enter the following command (converting different file formats is a common task):

```
samtools view -b -S -o mapped_reads.bam mapped_reads.sam
```

Parameters:

- b: output will be BAM format
- S: input is SAM format
- o: name of the output file

Next we **sort** the mapped reads by their genomic location and **index** them to allow efficient access:

```
samtools sort -o mapped_reads_sorted.bam mapped_reads.bam
samtools index mapped_reads_sorted.bam
```

Open *tablet* to visualize the location of the mapped reads on the reference genome. Choose **Open Assembly** and select the BAM file that you sorted and indexed in the last step (the initial BAM file will not work!). The mapped reads should show two genetic variants in comparison to the reference genome. Try to find them by visual inspection (tablet offers visualizations to help with this task).

**Question 3:** Enter the genomic position and type of the variants:

Genomic Location: **9345**, Base change from **A** to **T** (uppercase bases) (variant that comes first)

Genomic Location: **9497**, Base change from **A** to **G** (uppercase bases) (variant that comes second)

**Question 4:** Choose Open Features and select the GFF file (General Feature Format) to load the genome annotation (e.g. genes). On which gene are the variants located. Enter the gene name (match upper- and lowercase exactly):  
groES

## Assignment 03: Python I

### 1. Variables (1,5 Punkte)

One basic concept of all programming languages are **variables** - they allow you to store and retrieve values. Take a look at the following Python programs and try to predict the behavior of each program just by looking at the code. Afterwards you can copy the program, save it as a \*.py file and execute it with the Python interpreter (e.g. enter **python variables1.py** on the command line).

#### variables1.py

```
x = 1
x = x * 2
print x
```

What will this Python program print (enter **Error** if this is not a valid Python program): 2

#### variables2.py

```
my-number = 5
result = 5 - my-number
print result
```

What will this Python program print (enter **Error** if this is not a valid Python program): Error

#### variables3.py

```
x = 5
y = 4
y = y + 1
k = x * y
print k
```

What will this Python program print (enter **Error** if this is not a valid Python program): 25

## 2. Structured Programming (2,5 Punkte)

Usually lines of Python codes are executed sequentially from top to bottom. To create more dynamic programs you will use the following control structures:

- **conditional** statements (if / else)
- **repeated** statements (for loop, while loop)

These statements are the basic building blocks of **structured programming** languages and even complicated programs are just combinations of sequential, conditional, and repeated statements. If statements and for loops are present in all modern programming languages and they will appear in all but the most simple Python programs. **Please make sure that you fully understand their behavior before completing this assignment.**

Try to predict the behavior of each program just by looking at the code. Afterwards you should write the program, save it as a \*.py file and execute it with the Python interpreter.

**conditional.py:** How to use a simple conditional statement. Each if statement has a block that is only executed if a certain condition is met. It may have an else-block that is executed otherwise.

```
x = 0
for i in range(5):
    x = x + i
print x
```

What will the program conditional.py print: If-Block

**loop.py:** How to repeat statements with a for loop. Each for loop has a block that is executed for a given number of times. In order to answer this task you must understand two things:

1. How often will the for loop run (i.e. how often will it execute the indendent statement(s))
2. Which numbers will be summed up (i.e. which value will the loop variable *i* have during each iteration).

```
i = 2

if i <= 2:
    res = "If-Block"
else:
    res = "Else-Block"

print res
```

What will the program loop.py print: 10

How many times is the for loop executed (enter just a number, e.g. 1): 5

Which value has the loop variable *i* during the first iteration: 0

**structured.py:** You can mix both statements in any way, e.g. you can put an if statement within a for loop. Try to understand how this changes the result from the previous task:

```
x = 0
for i in range(5):
    if i != 2:
        x = x + i
print x
```

What will the program structured.py print: 8

### 3. Working with Strings (2 Punkte)

Every variable in Python has a **data type**. This data type is not explicitly stated, instead it is determined by the initial value of the variable and may change of the course of a program (you will learn much more about this in the next lecture). For this assignment it is sufficient to understand, that **Strings** (= *strings of characters*) are one of these data types. Strings are always **enclosed by single-quotes or double-quotes** (and forgetting to use single- or double quotes is one of the most common mistakes):

```
s = "This is a string"
```

Important operations for **Strings** are the selection of individual characters or the selection of substrings, prefixes, and suffixes. Complete the command such that the substring "chem" (without the double quotes) is printed.

```
A = "Biochemistry"
print
A[3:7]
```

Answer the following questions for yourself

What is the index of the first character in a string? 0

If you select a substring [a:b] is the character given by the index a included in the resulting substring? Included

If you select a substring [a:b] is the character given by the index b included in the resulting substring? Not included

### 4. Range function (1 Punkt)

The range function is often used in conjunction with for loops. Therefore it is very important to understand what the range function does. If you are unsure, enter help(range) in Python or consult the Python documentation found here (<https://docs.python.org/2/library/functions.html>):

The **len** function is often used in conjunction with **range**. It will simply return the length of a given string, e.g. this program will print: This string has length 4.

```
s = "ACGT"
print "This string has length", len(s)
```

In the following Example, what will the **range** function return?

```
A = "Test"
print range(len(A))
```

- ☐ range will return a list that contains the numbers 1, 2, 3, and 4.
- ☐ range will return all characters of the variable A in order (i.e. "T", "e", "s", "t").
- ☒ range will return a list that contains the numbers 0, 1, 2, and 3.
- ☐ range will return the length of the variable A (i.e. 4).

5. **For loops and strings** (2 Punkte)

You can use for loops to iterate over the characters in a string. To do this you need to understand:

1. for loops
2. the range function, the len function
3. how to select a character in a string by its index

Remember what the range function does. Which values will the loop variable *i* have over the course of the program? Now complete the following Python program that will **print every occurrence of the character A in the string s**. Line 3 is partially given, fill in the missing part so that the program works as intended:

```
s = "ACGTACGTA"
for i in range(len(s)):
    if
        s[i] == "A"

:
    print "I just saw the character A at index", i
```

6. **Bringing it all together** (3 Punkte)

In this task we will combine everything that you have learned in this assignment so far. In addition this task will show you some of the most common mistakes so that you are able to avoid them. Examine the following Python program:

```
seq = ACGTACGT
countA = 0

for i in range(seq)
    if seq(i) = A
        countA = countA + 1

print "There are", countA, "occurences of A in", seq
```

This program is supposed to count how often the character 'A' occurs in the string seq. Unfortunately there are mistakes in some of the lines. Correct the given lines such that the program works as intended:

```
seq = "ACGTACGT"

countA = 0

for i in range(len(seq)):
    if seq[i] == "A":
        countA = countA + 1

print "There are", countA, "occurrences of A in", seq
```

To test your program, copy it to a new file count.py and run it by entering python count.py.

## Assignment 04: Python II / Proteomics

### 1. Start Codon (2 Punkte)

In the last assignment you learned how to look for occurrences of the base A in a DNA or RNA sequence. In this task we will modify that program to look for the most common **start codon AUG** instead. Take a look at the following short mRNA sequence:

ACCAAUGAUAGCCGAUGGGUGAGGAG

We don't know the correct reading frame so we have to check for all overlapping triplets of bases whether they are the start codon or not. In this example we would first check ACC, then CCA, then CAA, and so on. We will continue in this way for the entire sequence and report all potential start codons. In this example you should find two start codons (as usual your program should still work if you replace that sequence with another one).

To solve this task remember

- Which value will the loop variable *i* have for each iteration of the for loop
- How can you use the loop variable *i* to access a single character in a string
- How can you select a substring

Complete the following program to **print the position of all start codons** in a given sequence seq:

```
seq = "ACCAAUGAUAGCCGAUGGGUGAGGAG"

for i in range(
    len(seq)-3
):
    if
        seq[i:i+3] == "AUG":

        print "AUG found starting at index", i
```

## 2. Working with Lists (1 Punkt)

In the lecture you learned about **lists** which are another important **data type** in Python. They are used to store a **collection of related data**, e.g. the sequences of a set of genes. In some ways lists are similar to strings, e.g. you can iterate over the **elements in a list** in the same way you iterate over the characters in a string. Look at the following Python program and try to predict its output just by looking at the code. **Test your solution afterwards.**

```
l = ['10', '20', '30', '40']
my_sum = l[0]

for i in range(len(l)):
    my_sum = my_sum + l[i]

print my_sum
```

Please enter the **exact** output of the command `print my_sum` at the end of the program:  
1010203040

## 3. List of strings (2 Punkte)

In the lecture you learned about **lists**, which are in some ways similar to **strings**. For example you can iterate over the **elements in a list** in the same way in which you iterate over the characters in a string:

```
l = ["ACGT", "CGTA", "TAAC", "TCGA"]

for i in range(len(l)):
    print l[i]
```

In this example the elements stored in the list are strings. A **list of strings** is something that you will see in many programs since you work with DNA or protein sequences most of the time.

Your task is to **count the total number of times the base A occurs in all strings in the list**, e.g. in the example above the correct solution is  $1+1+2+1 = 5$ . In order to count something in Python you need a variable that is updated every time you see the element that you want to count. Take a look at task 5 on the last assignment to see an example for this. Complete the following program:

```

l = ["ACGT", "CGTA", "TAAC", "TCGA"]

# Variable that will count how often we have seen the base A
# We have to initialize it before the for loop starts. Do you know why?
countA = 0

for i in range(len(l)):
    current_sequence =
    l[i]

    for j in range(len(current_sequence)):
        if
        current_sequence[j] == "A": oder l[i][j] == "A":

        countA = countA + 1

print "The base A occurs", countA, "times in all strings"

```

It is highly recommended to save your program as a file list.py and execute it by entering python list.py instead of using the interactive interpreter.

#### 4. Tryptic peptides (3 Punkte)

As discussed in the lecture, **shotgun proteomics**, also known as **bottom-up proteomics**, is the predominant technique used in laboratories. In **bottom-up proteomics**, proteins are first digested into peptides by an enzyme, then separated using a chromatographic column, ionized (most commonly using ESI) and analyzed in the mass spectrometer. **Trypsin** is the most commonly used enzyme to digest proteins into peptides for bottom-up proteomics. It hydrolyses amino acid chains at the carboxyl side of arginine and lysine (i.e. it cuts after R or K). Assuming you want to analyze the following protein (<http://www.uniprot.org/uniprot/Q9Y6D0>):

```

>sp|Q9Y6D0|SELK_HUMAN Selenoprotein K OS=Homo sapiens GN=SELK PE=1 SV=3
MVYISNGQVLDSRSQSPWRSLITDFFWGIAEFVVLFFKTLQDVKKRRSYGNSSDSRY
DDGRGPPGNPPRRMGRINHLRGPSPPPMAGGUGR

```

Which peptides do you expect in your sample if you digest this protein with trypsin? Complete the following Python program to answer this question. Your solution should work with any other protein sequence as well. As described above, trypsin cuts after R or K. We introduce **boolean logic (and, or, ...)** to model this behavior in Python:

```
s = "R"
if s == "R" or s == "K":
    print "s is either R or K"

i = 2
if i >= 1 and i <= 3:
    print "i is a number between 1 and 3"
```

It is highly recommended to create a file *trypsin.py*. Copy the following program and complete it. Execute it by entering *python trypsin.py*.

```
# The backspace character allows the string to continue on the next line
seq = "MVYISNGQVLDSRSQSPWRLSLITDFFWGIAEFVVLFFKTLQQDVKKRRSYGNSSDSRY" \
      "DDGRGPPGNPPRRMGRINHLRGPSPPPMAGGUGR"

# Index of the last cut that trypsin made
last_cut = 0

# List of all peptides that we will measure after trypsin digestion
# (= final result of the program)
peptides = []

# Iterate over the entire sequence. Once we encounter an amino acid
# R or K we build a new peptide (= substring from the last cut up to
# the current position) and add it to the list
for i in range(len(seq)):
    if seq[i] == "R" or seq[i] == "K":
        # Encountered R or K -> a peptide ends here
        # Get a substring of seq from the last_cut to the current position
        # ---
        # To see if your peptide is correct you should check that:
        # 1) The amino acid R or K is present the end of the peptide
        # 2) The amino acid R or K from the previous peptide is _not_
        #    present at the beginning of the next peptide
        # 3) The first amino acid (M) is present at the beginning of
        #    first peptide
        peptide =
seq[last_cut:i+1]

        # We only store peptides that are at least 4 amino acids long
        if len(peptide) >= 4:

peptides.append(peptide)

        last_cut = i + 1

print peptides
```

#### 5. Peptide Identification (4 Punkte)

In shotgun proteomics, we measure the mass-over-charge ratio ( $m/z$ ) of peptides. For complex mixtures this is not sufficient to identify the peptides. We therefore record MS/MS or **tandem mass spectra** that fragment selected peptides. Peptides fragment primarily at the amide bond between amino acids which leads to a **series of b- and y-ions** (ion ladders). The mass difference between consecutive b- or y-ions corresponds to the **residue mass** of amino acids and can be used to sequence the peptide.

The residue mass of an amino acid is its mass within a peptide, i.e. the mass of the amino acid minus the mass of water. In addition you have to add +H (~ +1 Da) at the N-terminal end (b-ions) and +H<sub>2</sub>O + H (~ +19 Da) at the C-terminal end (y-ions) of the protein. Since tryptic peptides always end with lysine or arginine you expect characteristic masses for the y<sub>1</sub> ion, either m/z 147 for lysine or m/z 175 for arginine.

You are given a tandem mass spectrum containing the following masses (m/z values). The precursor mass of the spectrum is 634.36 Da (at charge 1).

88.04  
147.11  
260.20  
274.12  
361.25  
375.17  
488.25  
547.32  
616.35  
634.36

Make the following simplified assumptions (actual tandem mass spectra will contain much more peaks):

- The spectrum contains only b- and y-ions (and the precursor mass)
- No b- or y-ions are missing from the spectrum
- All m/z values correspond to singly positively charged, monoisotopic ions
- The instrument has an accuracy of +/- 0.5 Da

Identify the peptide sequence from the spectrum. To do this you have to find the b- and/or y-ion ladder in the spectrum. Use the following table of amino acid residue masses to identify the peptide sequence from the distances between consecutive b- or y-ions: [http://www.matrixscience.com/help/aa\\_help.html](http://www.matrixscience.com/help/aa_help.html)

What is the mass of the b<sub>1</sub> ion: 88.04

What is the mass of the y<sub>1</sub> ion: 147.11

Identify the complete peptide sequence (5 amino acids): SWTIK oder SWTLK (Serine, Tryptophan, Threonine, (Iso-) Leucine, Lysine)

## Assignment 05: Python III

### 1. A first function (1 Punkt)

You have a large **collection of DNA and RNA sequences**. Now you need to distinguish which are DNA sequences and which are RNA sequences. Your colleague wrote the following function to **check whether a given sequence contains the character U or not** (if it contains a U it must be a RNA sequence).

You test his/her function with the two example sequences **ATCG** and **AUCG**. Try to predict the output just by looking at the code, try to implement the function afterwards. Can you explain the result?

```
def is_RNA_sequence(seq):
    found_U = False

    for i in range(len(seq)):
        if seq[i] == "U":
            found_U = True
        else:
            found_U = False

    return found_U

print is_RNA_sequence("ATCG"), is_RNA_sequence("AUCG")
```

- ☐ True True
- ☐ True False
- ☐ False True
- ☒ False False

## 2. Prime numbers (3 Punkte)

You want to write a **function that checks whether a given number is prime or not** (i.e. a number that is only divisible by 1 and itself). We will use a very simple approach that divides the given number by every number that is smaller except 0 and 1 (division by 0 is not possible and division by 1 will always have a remainder of 0). **If the remainder of any of these divisions is 0, we know that the given number cannot be a prime number.** Use the operator % to get the **remainder** of a division (one example for this is  $14 / 4 = 3$  and  $14 \% 4 = 2$ . Another example is  $16 / 4 = 4$  and  $16 \% 4 = 0$ ).

Use **range(start, stop)** to determine the beginning and the end of the list returned by *range()*. *start* is contained in the resulting list, *end* is not contained in the resulting list, e.g. *range(4, 7)* will result in *[4, 5, 6]*.

```
def is_prime(x):
    # We assume that x is prime until we find a number
    # that is divisible without remainder
    prime = True

    # Divide x by every number that is smaller (except 0 and 1)
    # If the remainder of any division is 0, we know that x can
    not
    # be a prime number.
    for i in range(
        2
        ,
        x oder x/2
    ):
        if x % i == 0:
            prime = False

    return prime
```

You can now use your function to find all prime numbers in a given range of numbers. For example you could test your function like this:

```
print "Here are all prime numbers between 2 and 100:"
for i in range(2, 101):
    if is_prime(i):
        print i
```

3. **In order to understand recursion, one must first understand recursion. With factorials.** (2 Punkte)

Some problems can be written in a simple, recursive manner. Recursion means that a function will call itself. Each time the function is called with a smaller problem, until the problem is trivial to solve. This so-called **base case** is very important because otherwise the function would never stop to call itself. Once the base case is reached, the function returns a value for the first time. This value is now used to solve the previous, slightly bigger problem and so on. You want to implement the **factorial function** which can be defined recursively as

$$n! = \begin{cases} 1 & \text{if } n = 0, \\ (n-1)! \times n & \text{if } n > 0 \end{cases}$$

Visualize the recursive function calls and their parameters and return values on a sheet of paper. Fill in the missing parts below:

```
def fac(n):
    # Base case -> trivial solution, recursion will stop.
    if n == 0:
        return
    1

    # Recursion -> solve smaller version of the original problem
    else:
        return n*fac(n-1)
```

Test your solution by comparing it to the function provided by Python:

```
import math
print "My function", fac(5)
print "Python function", math.factorial(2)
```

4. **Nested for loops** (2 Punkte)

For many tasks you will need **nested for loops** (= a for loop within a for loop). For example you could use them to compute the distance between **all pairwise combinations** of the sequences in a list (sometimes, like and this and the next task, you need to exclude some combinations). For this task you are given the following lists l1 and l2:

```
l1 = ["A", "B", "C", "D"]
l2 = ["1", "2", "3", "4"]
```

```

for i in range(len(l1)):
    for j in range(
        i
    ,
        len(l2)
    ):
        print l1[
            i
        ] + l2[
            j
        ]

```

such that they will produce the following output:

A1  
 A2  
 A3  
 A4  
 B2  
 B3  
 B4  
 C3  
 C4  
 D4

**Hint:** In the inner for loop you must use *range(start, stop)* to determine the beginning and the end of the list returned by *range()*. *start* is contained in the resulting list, *end* is not contained in the resulting list, e.g. *range(4, 7)* will result in *[4, 5, 6]*.

#### 5. Similarity between sequences (4 Punkte)

You are given a list of sequences of equal length. Your task is to find the two sequences that are most similar given the following **similarity function** for nucleotides:

$$\text{similarity}(a, b) = \begin{cases} 1, & \text{if } a = b \\ 0.5, & \text{if } a \neq b, a \text{ and } b \text{ are both either purines or pyrimidines} \\ 0, & \text{if } a \neq b, a \text{ and } b \text{ are not both either purines or pyrimidines} \end{cases}$$

We will compare entire sequences by using a simple nucleotide-wise comparison like this:

ATCCGGT  
 GCGTTAC

= 0.5 + 0.5 + 0 + 0.5 + 0 + 0.5 + 0.5 = **Similarity of 2.5**

Usually you would need to align the sequences first in order to compare them. You will learn about that in the next lecture. You could implement a similarity function for complete sequences like shown below (this is just one way to do it, there are many different correct solutions). This implementation checks whether the current nucleotide is present in the list of purines or in the list of pyrimidines.

In Python you use the keyword **in** to check whether an element occurs in a list (or a substring occurs in a string). For example:

```
#####
# Lists
#####
l1 = ["A", "G"]
print "A" in l1    # Will print True
print "C" in l1    # Will print False

# Also works with longer strings
l2 = ["ACGT", "CTTA"]
print "ACGT" in l2  # Will print True

#####
# Strings
#####
start_codon = "AUG"
seq = "ATCGAUGCTGA"

print start_codon in seq # Will print True
print "ABC" in seq       # Will print False
```

Complete the implementation of the similarity function to compare two sequences a and b:

```
# This function calculates the similarity between two sequences a and b
def similarity(a, b):
    purine = ["A", "G"]
    pyrimidine = ["T", "C"]

    if len(a) != len(b):
        print "Error: both sequences must have the same length."
        return None

    result = 0

    for i in range(len(a)):
        if a[i] == b[i]:
            result = result +
1
        elif a[i] in purine and b[i] in purine:
            result = result +
0.5
        elif a[i] in pyrimidine and b[i] in pyrimidine:
            result = result +
0.5
        else:
            result = result +
0
    return result
```

Use this function to find the two most similar sequences in the given list of sequences like this:

```
sequences = ["ATCCGGT", "GCGTTAC", "CTACTGC", "TTGCAGT", "AGTCACC"]

# Initialize variables
highest_similarity = 0
index1 = 0
index2 = 0
```

You need to consider **all pairwise comparisons between the sequences** in the list to find the two sequences with the highest similarity (this requires a nested for loop). We **store the highest similarity that we have seen so far** (starting with 0) and update it as soon as we see a higher similarity between two sequences. After all pairwise comparisons this variable will contain the highest similarity between any two sequences, but it won't tell us which

sequences had this similarity. Therefore we need two additional variables index1 and index2 to **store the indices of the two sequences that had the highest similarity so far**.

Important: Make sure that you don't compare a sequence against itself in the inner for loop. They will have the highest similarity, but of course that's not what we are searching for.

```
# Find the two most similar sequences.
# Make sure that you don't compare a sequence
# against itself!
for i in range(len(sequences)):
    for j in range(
        i+1
    ,
        len(sequences)
    ):
        s = similarity(
            sequences[i]
        ,
            sequences[j]
        )
        if s > highest_similarity:
            highest_similarity =
                s
            index1 = i
            index2 = j

print "The two sequences with the highest similarity are"
print sequences[index1], "and", sequences[index2]
print "Their similarity is", highest_similarity
```

## Assignment 06: Alignment I

### 1. Needleman-Wunsch score (0,75 Punkte)

Where can you find the score of the optimal alignment after you filled in the dynamic programming matrix with the Needleman-Wunsch algorithm?

- ☐ Always in the top left corner
- ☐ Always in the top right corner
- ☐ Always in the bottom left corner
- ☒ Always in the bottom right corner
- ☐ The lowest score in the dynamic programming matrix
- ☐ The lowest score in the dynamic programming matrix
- ☐ The alignment score is marked with the value "1" in the traceback matrix
- ☐ The alignment score is marked with the value "0" in the traceback matrix

2. **Needleman-Wunsch** (1,5 Punkte)

Select **all** statements regarding the Needleman-Wunsch algorithm that are correct:

- ☐ You need the traceback matrix to find the optimal alignment score
- ☒ After initialization you can fill out the dynamic programming matrix row by row or column by column
- ☒ Needleman-Wunsch will always find the optimal alignment
- ☐ Needleman-Wunsch has linear runtime
- ☐ The traceback stops once you have reached a cell with the value 0 in the traceback matrix

3. **Needleman Wunsch Matrices** [Variante 3] (3 Punkte)

You want to calculate the optimal alignment between the sequences **A=GAAGGCT** and **B=GATGGAACC** using the Needleman-Wunsch algorithm. Use the following distance function and a gap cost of 1.

$$distance(a, b) = \begin{cases} 0, & \text{if } a = b \\ 0.5, & \text{if } a \neq b, a \text{ and } b \text{ are both purine or pyrimidine bases} \\ 1, & \text{if } a \neq b, a \text{ and } b \text{ are not both purine or pyrimidine bases} \end{cases}$$

**Dynamic programming matrix**

|   | - | G | A | T   | G   | G   | A | A | C | C |
|---|---|---|---|-----|-----|-----|---|---|---|---|
| - | 0 | 1 | 2 | 3   | 4   | 5   | 6 | 7 | 8 | 9 |
| G | 1 | 0 | 1 | 2   | 3   | 4   | 5 | 6 | 7 | 8 |
| A | 2 | 1 | 0 | 1   | 2   | 3   | 4 | 5 | 6 | 7 |
| A | 3 | 2 | 1 | 1   | 1.5 | 2.5 | 3 | 4 | 5 | 6 |
| G | 4 | 3 | 2 | 2   | ?   |     |   |   |   |   |
| G | 5 | 4 | 3 | 3   |     |     |   |   |   |   |
| C | 6 | 5 | 4 | 3.5 |     |     |   |   |   |   |
| T | 7 | 6 | 5 | 4   |     |     |   |   |   |   |

**Traceback matrix**

From which cell was the score calculated, diagonal (match bases) = 0, left (add gap in A) = 1, up (add gap in B) = -1.

|   | -  | G  | A  | T | G | G | A | A | C | C |
|---|----|----|----|---|---|---|---|---|---|---|
| - | 1  | 1  | 1  | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| G | -1 | 0  | 1  | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| A | -1 | -1 | 0  | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| A | -1 | -1 | 0  | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| G | -1 | 0  | -1 | 0 | ? |   |   |   |   |   |
| G | -1 | 0  | -1 | 0 |   |   |   |   |   |   |
| C | -1 | -1 | -1 | 0 |   |   |   |   |   |   |
| T | -1 | -1 | -1 | 0 |   |   |   |   |   |   |

Which value do you enter into the cell marked in red in the **dynamic programming matrix**: 1

Which value do you enter into the cell marked in red in the **traceback matrix**: 0

4. **Traceback Matrix** [Variante 3] (3 Punkte)

The following traceback matrix was generated by Needleman-Wunsch for the sequences **A = CGCAAGGC**, **B = AAGCAATTGACA**. Reconstruct the optimal alignment from the traceback matrix and enter the sequences including the gaps (use - for gap). As usual the traceback matrix was created using diagonal (match bases) = 0, left (add gap in A) = 1, up (add gap in B) = -1.

|   | -  | A  | A  | G  | C  | A  | A  | T | T | G | A | C | A |
|---|----|----|----|----|----|----|----|---|---|---|---|---|---|
| - | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1 | 1 | 1 | 1 | 1 | 1 |
| C | -1 | 0  | 0  | 0  | 0  | 1  | 1  | 1 | 1 | 1 | 1 | 0 | 1 |
| G | -1 | 0  | 0  | 0  | 1  | 0  | 0  | 1 | 1 | 0 | 1 | 1 | 1 |
| C | -1 | -1 | 0  | 0  | 0  | 1  | 1  | 0 | 0 | 1 | 0 | 0 | 1 |
| A | -1 | 0  | 0  | 0  | -1 | 0  | 0  | 1 | 1 | 1 | 0 | 1 | 0 |
| A | -1 | 0  | 0  | 0  | 0  | 0  | 0  | 1 | 1 | 1 | 0 | 1 | 0 |
| G | -1 | -1 | -1 | 0  | 0  | -1 | -1 | 0 | 0 | 0 | 1 | 1 | 1 |
| G | -1 | -1 | -1 | 0  | 0  | 0  | -1 | 0 | 0 | 0 | 0 | 1 | 0 |
| C | -1 | -1 | -1 | -1 | 0  | 0  | -1 | 0 | 0 | 0 | 0 | 0 | 1 |

Please note that sequence A is on the left side of the matrix.

A: -CGCAA--GGC-

B: AAGCAATTGACA

5. **Needleman-Wunsch in Python** [Variante 1] (3,75 Punkte)

The following program is an implementation of the Needleman-Wunsch algorithm in Python (for your convenience the program is also linked above as a .py file). Study it closely and complete the five incomplete lines. Execute the program once you are done. If everything is implemented correctly you will see all pairwise alignment for the sequences in the list I.

Which alignment between any two sequences in the list I has the smallest distance. Enter the aligned sequences **with** gaps:

Sequence 1: GATTTTATACA

Sequence 2: GAGCTCAT--A

Distance: 5.0

```

# Import matrix class from numpy
import numpy

# A simple distance function: count the mismatches
def d(a,b):
    if a == b:
        return 0
    else:
        return 1

def NW(A, B):
    # Length of the sequences
    m = len(A)
    n = len(B)

    # Initialize the DP matrix and the traceback matrix
    # Both have the dimension (m+1) x (n+1)
    # Task for yourself: Explain why the DP and traceback
    # matrix have the dimension (m+1) x (n+1).
    D = numpy.zeros([m+1, n+1])
    T = numpy.zeros([m+1, n+1])

    # Initialize the first row and the first column
    # of both matrices
    for i in range(m+1):
        D[i, 0] = i * d("-", "A")
    for j in range(n+1):
        D[0, j] = j * d("-", "A")
    for i in range(m+1):
        T[i,0] = -1
    for j in range(n+1):
        T[0,j] = 1

    # Dynamic programming: fill in the DP matrix
    for i in range(1,m+1):
        for j in range(1,n+1):
            # Calculate the distances
            d_horiz = D[i-1,j] + d(A[i-1],"-")
            d_diag =
D[i-1,j-1] + d(A[i-1],B[j-1])

            d_vert =
D[i,j-1] + d("-",B[j-1])

            # Save the smallest distance in the DP matrix
            D[i,j] =
min

([d_horiz, d_diag, d_vert])

            # Remember the direction in the traceback matrix
            if d_diag <= d_horiz and d_diag <= d_vert:
                T[i,j] = 0
            elif d_horiz <= d_vert:
                T[i,j] = -1
            else:
                T[i,j] = +1

```

```

# The aligned sequences (with gaps)
# will be called X and Y (A' and B'
# are not valid variable names).
X = ""
Y = ""

# Traceback: start in the bottom right cell
# of the matrix and continue until you reach
# the cell at the top left.
i = m
j = n
while i > 0 or j > 0:
    if T[i,j] ==
0
: # Diagonal: no gaps
    X = A[i-1] + X
    Y = B[j-1] + Y
    (i,j) = (i-1, j-1)
    elif T[i,j] ==
-1
: # To the top: gap in B
    X =
A[i-1]
+ X
    Y = "-" + Y
    i = i - 1
    else: # To the left: gap in A
    X = "-" + X
    Y =
B[j-1]
+ Y
    j = j - 1

    return D[
len(A) oder m
,
len(B) oder n
], X, Y

l = ['ACGTATGTAC', 'GATTTTATACA', 'GAGCTCATA', 'ATGCGATGAGATAG', 'CATTGTGAGCT
']

for i in range(len(l)):
    for j in range(i+1, len(l)):
        dist, X, Y = NW(l[i], l[j])
        print X
        print Y
        print "Distance:", dist
        print ""

```

# Assignment 07: Alignment II / MSA

## 1. PAM vs. BLOSUM (1 Punkt)

There are some differences between the commonly used **BLOSUM** and **PAM** substitution matrices. This also manifests in the naming scheme of the matrices, e.g. BLOSUM62 and PAM120. It is important to **select the right scoring matrix for different levels of sequence identity** (i.e. evolutionary distance). Please choose **all** correct statements.

- ☒ Higher PAM numbers denote larger evolutionary distance
- ☐ Higher PAM numbers denote higher sequence similarity
- ☐ Lower BLOSUM numbers denote higher sequence similarity
- ☒ Lower BLOSUM numbers denote larger evolutionary distance

## 2. Fill the Dynamic Programming Matrix [Variante 1] (2 Punkte)

Fill the complete dynamic programming matrix using **Needleman-Wunsch** and the following simple **distance function**

$$\text{dist}(a, b) = \begin{cases} 0 & \text{if } a == b, \\ 1 & \text{else} \end{cases} \text{ (i.e., gap cost = 1).}$$

Bonus question: Fill in the traceback matrix as well and write down the aligned sequences.

|   | - | C | T | A |
|---|---|---|---|---|
| - | 0 | 1 | 2 | 3 |
| C | 1 | 0 | 1 | 2 |
| A | 2 | 1 | 1 | 1 |
| T | 3 | 2 | 1 | 2 |
| G | 4 | 3 | 2 | 2 |

## 3. Needleman-Wunsch with Similarity Function [Variante 2] (2,5 Punkte)

In the last assignment you learned how to fill in the Needleman-Wunsch dynamic programming matrix using a distance function. How does this change when you use a **similarity function** instead?

First, how do you initialize the first row and the first column of the dynamic programming matrix, when you are using the following **similarity function**:

$$\text{similarity}(a, b) = \begin{cases} 0, & \text{if } a \neq b \neq \epsilon \\ 1, & \text{if } a = b \\ -2, & \text{if } a = \epsilon \text{ or } b = \epsilon \end{cases}$$

In the traceback matrix use 0 for diagonal, 1 for left and -1 for up.

**Dynamic programming matrix**

|   | -  | A | T | G  |
|---|----|---|---|----|
| - | 0  | ? | ? | A2 |
| T | ?  |   |   |    |
| A | ?  |   |   |    |
| G | ?  |   |   |    |
| T | A1 |   |   |    |

**Traceback matrix**

|   | -  | A | T | G  |
|---|----|---|---|----|
| - | 0  | ? | ? | A4 |
| T | ?  |   |   |    |
| A | ?  |   |   |    |
| G | ?  |   |   |    |
| T | A3 |   |   |    |

Find the correct values for all cells in the first row and the first column in the dynamic programming matrix and the traceback matrix and enter the values for the fields marked with A1, A2, A3 and A4 in the respective text fields.

A1: -8, A2: -6, A3: -1, A4: 1

And the function for  $D_{i,j}$  is:

$$D_{i,j} = \max \begin{cases} D_{i,j-1} + d(a_i, \epsilon) \\ D_{i-1,j-1} + d(a_i, b_j) \\ D_{i-1,j} + d(\epsilon, b_j) \end{cases}$$

4. **Smith-Waterman Multiple Choice** (1 Punkt)

Select **all** statements that are true if you use the Smith-Waterman algorithm with a similarity function.

- ☒ The traceback starts at the cell that has the highest score in the dynamic programming matrix
- ☐ The traceback starts at the cell that has the highest score in the traceback matrix
- ☒ The traceback stops once it has reached a cell that has the value 0 in the dynamic programming matrix
- ☐ The traceback stops once it has reached a cell that has the value 0 in the traceback matrix
- ☐ The alignments calculated with Smith-Waterman will not contain gaps
- ☒ The dynamic programming matrix cannot contain negative values
- ☐ The traceback matrix cannot contain negative values
- ☒ An alignment can begin or end with a mismatch

5. **Smith-Waterman** [Variante 3] (3 Punkte)

Find the optimal **local alignment** using Smith-Waterman for **A = CATAC** and **B = AGTACC**.

Use the following **similarity function** (i.e. +2 for a match, -1 for a mismatch) and a gap cost of -1:

$$\text{similarity}(a, b) = \begin{cases} 2, & \text{if } a = b \\ -1, & \text{if } a \neq b \end{cases}$$

Fill in the **dynamic programming matrix** and the traceback matrix (not shown). Enter the value of the fields A1, A2 and A3 in the **dynamic programming matrix**. A1: 0, A2: 3, A3: 6

What is the **score** of the optimal local alignment: 7  
Enter the optimal local alignment including gaps. A: A-TAC, B: AGTAC

|   | - | A  | G | T  | A | C | C  |
|---|---|----|---|----|---|---|----|
| - |   |    |   |    |   |   |    |
| C |   | A1 |   |    |   |   |    |
| A |   |    |   |    |   |   |    |
| T |   |    |   | A2 |   |   |    |
| A |   |    |   |    |   |   |    |
| C |   |    |   |    |   |   | A3 |

## 6. From Proteins Databases to Multiple Sequence Alignments (2,5 Punkte)

Multiple sequence alignments (MSAs) are useful to identify conserved regions in related proteins. We assume that conserved regions are crucial for the structure and/or the function of the protein. In this task we will investigate **lysozyme C** across different species. As discussed in the lecture, **Uniprot** ([uniprot.org](http://www.uniprot.org)) is one of the most commonly used protein databases. Use the search field to find the entry for **human lysozyme C**. **Take some time to look over the information that Uniprot provides** about this protein. In particular, you can find information about the active sites that are directly involved in catalysis.

Which amino acids of human lysozyme C are the active sites (enter their **position**): 53 and 71

Each Uniprot entry contains a button to retrieve the protein sequence in FASTA format. But **Uniprot makes it particularly easy to retrieve data with automated programs**, see [http://www.uniprot.org/help/programmatic\\_access](http://www.uniprot.org/help/programmatic_access)  
For example, you can retrieve the FASTA sequence for an entry like this: <http://www.uniprot.org/uniprot/P12345.fasta>

This means that **you can automatically download the protein sequences for any number of proteins** instead of visiting hundreds of pages to copy and paste the sequences one by one. In an earlier lecture, you learned about a Unix program that allows you to download files from remote sources. We use the Python function **os.system** to call this program from Python. The following Python program is supposed to automatically download a list of protein sequences from Uniprot. Finish it by entering the program introduced in the lecture:

```
import os

accession = ["P61626", "P00698", "P00697", "P17897", "Q7LZQ1", "P61629",
             "Q06283", "P79179", "P51782", "P16973", "P24364"]

for acc in accession:
    os.system("
wget
-P sequences http://www.uniprot.org/uniprot/" + acc + ".fasta")
```

**Note:** This program should be pre-installed on any Linux computer, but it is not pre-installed on Mac OS X or Windows. Therefore the Python program will not work on these systems.

1. The parameter **-P** is used to download all files to a directory called *sequences* which will be created in the current directory. Delete this directory if it already exists and execute the Python program.
2. We need to combine the individual *.fasta* files in a single *.fasta* file in order to compute the MSA. In an earlier lecture you learned how to combine the contents of multiple files in a single file. **Enter the full Unix command that combines the content of all downloaded .fasta files in a new file called sequences.fasta.** Assume that your current directory is the directory *sequences* which contains the downloaded *.fasta* files.  
`cat *.fasta > sequences.fasta`
3. The European Bioinformatics Institute (EBI) provides ClustalO (successor of ClustalW) as a web service: <http://www.ebi.ac.uk/Tools/msa/clustalo/>. We will use this to calculate the MSA of the downloaded lysozyme C sequences. Click "Browse" and upload your file *sequences.fasta*. Don't change any parameters and submit the job.

4. Take a look at the results (click "Show Colors"). You will see several **regions of fully or strongly conserved amino acids** (denoted by \* and :). The longest region of fully or strongly conserved amino acids should be located (roughly) in the middle of the protein sequence. Enter the first and last amino acid of this region using the one letter code. From **S** to **P**. Are the active sites fully conserved?
5. Click on "Phylogenetic Tree" to study the **evolutionary relationship between the sequences**. The lysozyme C of which species is most closely related to the human one according to the phylogenetic tree generated by ClustalO. Retrieve the **latin species name** from Uniprot (e.g. *Homo sapiens*):  
Gorilla gorilla gorilla  
oder Gorilla gorilla  
oder Gorilla gorilla gorilla (Western lowland gorilla)

## Assignment 08: BLAST

### 1. E-value (3 Punkte)

A very important part of a BLAST result is the **E-value**. What is it and what is it good for? Please select **all** correct statements. The E-value...

- ☐ is between zero and one.
- ☒ is between zero and infinity.
- ☐ is between one and infinity.
- ☒ depends on the database size.
- ☒ depends on the length of the query sequence.
- ☒ is the number alignments with score s or better you would expect to see by chance when searching the given database.
- ☐ is the probability that an alignment with score s was found by chance when searching the given database.

### 2. BLAST web interface (1 Punkt)

Visit the NCBI website (<http://www.ncbi.nlm.nih.gov>) and search for the Yersinia pestis toxin ymt in the protein database. On the result page select the reference sequence. In the database entry you will see several functions regarding sequence analysis on the right side. Choosing "Run BLAST" will forward you to the NCBI BLAST web interface. In this case the correct accession number is automatically inserted into the search field. Search this sequence against the UniProt database. Make sure that you use blastp.

**Make yourself familiar with the result page.** The second figure shows a distribution of the BLAST hits on the query sequence. What does the color coding and the length of the individual lines represent? Now look at the list of hits. Notice the ascending order of the results with respect to the E-value. Look at the alignment of the top result. What is its coverage, alignment score, E-value, and sequence identity?

Now look at the graphical representation of the search results that show the conserved domains. You should see two conserved domains. To which **superfamily** do they belong (SMART name or name used on the BLAST result page):

PLDc\_SF  
oder PLDc\_SF superfamily  
oder PLDc

### 3. BLAST on the command line (6 Punkte)

The NCBI BLAST web interface only allows searches against pre-defined databases, e.g. UniProt. You can restrict your BLAST search to a subset of the database by organism or ENTREZ query. If this is not sufficient, e.g. if you have your own database or if you need to perform a large number of individual BLAST queries, you can use the BLAST command line tool on your own computer.

First, download all protein sequences of *Arabidopsis thaliana* from NCBI (<http://www.ncbi.nlm.nih.gov>). To do this, select "Genome" and search for *Arabidopsis thaliana*. The genome page links all protein sequences in FASTA format. There are also protein files for individual chromosomes but we are not interested in those. Look near the top of the page for the files containing genome-wide data.

Use `makeblastdb` to create the index files. Enter **`makeblastdb -help`** to see a list of all parameters. The most important ones are:

```
-in      the FASTA database for which you want to generate an index
-dbtype  the database type (nucl, prot)
```

Afterwards you should see additional index files ending with .phr, .pin, and .psq next to your database file.

Many other plant genomes are not as well annotated as the *Arabidopsis thaliana* genome. The file **unknown\_protein.fasta** (linked above) contains the sequence of an unknown protein from a closely related species. Download the file and try to assign a function by using BLAST to find a homologous sequence in the genome of *A. thaliana*.

Which variant of BLAST do you need to use: `blastp`

To learn about the parameters of the BLAST command line tool use the parameter `-help`. The two parameters you need for your query are:

```
-db
    the database to search in (the one you created earlier)

-query
    the input file name (containing the DNA sequence you want to BLAST)
```

To which large protein family does the unknown rice gene belong according to your BLAST search: Cytochrome P450 (oder p450 oder p450 superfamily oder cytochrome P450, family 86, subfamily A oder Cytochrome P450 86A4)

In addition the file **miseq\_output.fasta** contains a DNA sequence from a recently sequenced plant genome that is not yet annotated. Which variant of BLAST do you need to use to search this sequence against your *A. thaliana* database: `blastx`

Enter the accession of the top BLAST search result: NP\_175088 (oder: NP\_175088.1)

#### 4. Biopython BLAST (2 Punkte)

In the previous task you used the BLAST command line tools to perform BLAST searches against the *A. thaliana* genome. Now we will automate this task using Python / Biopython.

**Please complete the task "BLAST on the command line" before working on this task.**

Download the linked Python program and try to understand it. You can use this program to BLAST **all** protein sequences given in the file *oryza.fasta* against the *A. thaliana* genome (this file contains a selected subset of *Oryza sativa* proteins). In this example we are only interested in the best BLAST result from any of the given sequences, but you could modify the program to automatically search for results that are interesting for your research question.

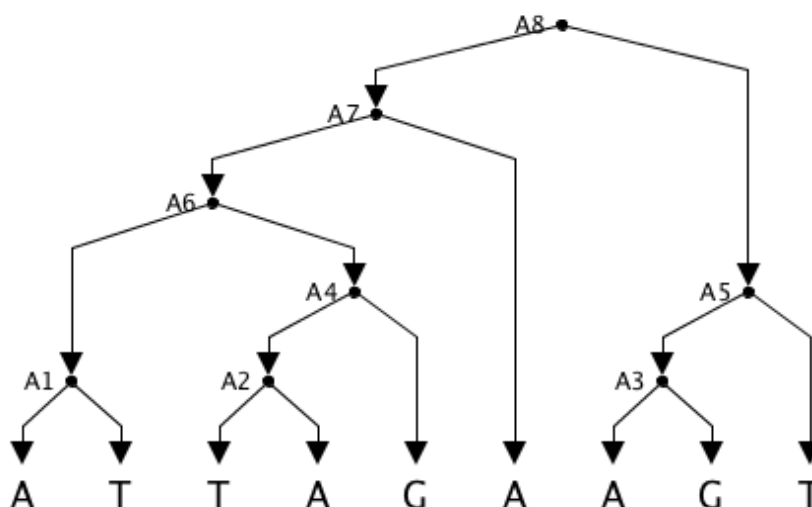
Call the program by executing **python annotate.py oryza.fasta** on the command line. Make sure that all files are in the current working directory or give their full relative or absolute path.

Enter the protein name of the top result given by *annotate.py*: Alba DNA/RNA-binding protein  
oder: Alba DNA/RNA-binding protein [Arabidopsis thaliana]

## Assignment 09: Phylogeny / Systems Biology

#### 1. Phylogeny - Small Parsimony Problem (2 Punkte)

The goal of phylogenetic methods is to find the true evolutionary tree that generated a set of known sequences (e.g. a column in a multiple sequence alignment). The **small parsimony** problem is easy to solve and finds the optimal labeling of a given tree such that the overall number of changes is as small as possible. A much more difficult problem is the large parsimony problem, which will also find the tree topology that will give the best score (i.e. smallest number of base changes in any possible tree topology). In this assignment we will only solve the small parsimony problem, that means the topology of the evolutionary tree is already known. Use **Fitch's algorithm** to label the ancestors in the following tree:



Fitch's algorithm consists of two phases, the bottom-up phase and the top-down phase: Fill in the labels after the **bottom-up phase**. Write sets of labels in alphabetical order and separate them using a comma (e.g. **A, C**)

A1: A, T  
A2: A, T  
A3: A, G  
A4: A, G, T  
A5: A, G, T  
A6: A, T  
A7: A  
A8: A

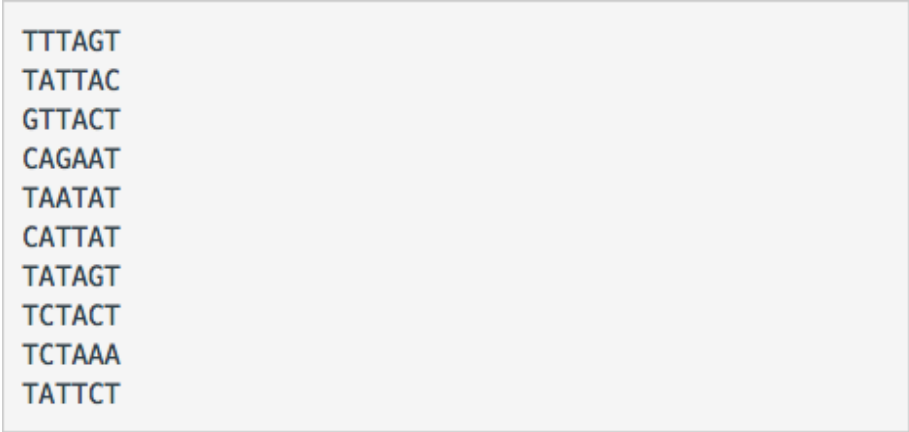
Complete the labels after the **top-down phase**:

A8: A, A7: A, A6: A, A5: A, A4: A, A3: A, A2: A, A1: A

## 2. **Sequence Logo** (3 Punkte)

TATA boxes are evolutionary well-conserved elements in the promotor region of many eukaryotic genes. As you already learned, multiple sequence alignment (MSAs) serve as basis for the construction of phylogenies. One elegant way to visualize the conservation in each column are sequence logos (first introduced in Lecture 07 - Alignment II / MSA). The height of each character corresponds to the information content of that character at that position. Highly conserved characters correspond to high information content.

Calculate the information content  $R$  for the given nucleotide sequences at each position (column). Please round each result to two digits after the decimal point (e.g.: 0.56).



TTTAGT  
TATTAC  
GTTACT  
CAGAAT  
TAATAT  
CATTAT  
TATAGT  
TCTACT  
TCTAAA  
TATTCT

0.84 | 0.63 | 1.08 | 1.03 | 0.51 | 1.08

You can compare your results to the figures that [weblogo](#) creates (change the setting "Composition" to "No adjustment for composition"). The information content  $R$  corresponds to the overall height at each position in the logo.

If you want to solve this with Python, you will find the  $\log_2$  calculation quite easy (<https://docs.python.org/2/library/math.html#power-and-logarithmic-functions>). Keep in mind that in Python  $2/3 = 0$  and  $2.0/3.0 = 0.66666666$ .

### 3. KEGG Pathways (3 Punkte)

In systems biology, we are interested in complex interactions in various biological systems. As discussed in the lecture, these interactions are represented by different kinds of biological networks. They are stored in a multitude of different databases available online. One of the most famous pathway databases is the [Kyoto Encyclopedia of Genes and Genomes \(KEGG\)](#). Take a look around and make yourself familiar with the website. As usual every database entry can be retrieved by an accession or entry number. Glycolysis is one of the oldest metabolic pathways. Try to find this pathway on KEGG. What is its entry number (e.g. map00001): map00010

Now take a closer look at the glycolysis pathway. What do the nodes and edges represent:

Nodes: Metabolites

Edges: Enzymes

The enzymes are depicted by EC numbers (Enzyme Commission). The first digit denotes the general reaction type. For example phosphoglucose isomerase has the code EC 5.4.2.2. Which general reaction type does this correspond to: Isomerases

In the lecture we discussed the circadian rhythm. Locate the circadian rhythm for plants in KEGG to answer the following questions:

Which photoreceptors are responsible for the response to red light and far red light?

Phytochrome A and Phytochrome B.

Which protein is the last step in the process leading to the induction of the flowering of the plant? Flowering Locus T (oder protein FLOWERING LOCUS T oder protein flowering locus T oder FLOWERING LOCUS T oder FT oder TSF).

### 4. Systems Biology (4 Punkte)

The article "Can a Biologist Fix a Radio? - or, What I Learned while Studying Apoptosis" (Cancer Cell, 2002) is a highly recommended discussion article that is both insightful and entertaining. You can find the article [here](#). In addition it is attached above. The author uses the analogy between a broken radio and, for example, human cancer cells and discusses how a systems biology approach can complement traditional experimental approaches. He states that listing all components of a radio/cell and describing their relationships can be successful to identify the cause of a disease. However, he also discusses why this approach is in many cases unable to do this. According to the author, what is the key reason that traditional experimental approaches such as listing all components of a radio/cell and describing their relationships may be insufficient to find the cause of a defect/disease in a radio/cell. Write a single sentence (you may copy a sentence from the paper if you want to).

[Punkte werden auf Basis des Vorkommens der folgenden Begriffe vergeben: - tunable components - several components are not tuned properly]

Bewertungsmodus: Automatische Bewertung bei Nennung eines Begriffs

# Assignment 10: Protein Structure / Secondary Structure

## 1. PDB (1,5 Punkte)

PDB (Protein Data Bank) is a database for the 3D structures of proteins (~90% XRD, ~9% NMR). Search for the entry with the accession number 4PTI. You should see a XRD structure of a protein. First, you should always check the resolution of the data. The resolution is: 1.5 Å.

Take a closer look at the 3D structure and the wealth of information that PDB provides. As discussed in the lecture, most databases, including PDB, are interlinked. Give the Uniprot accession for this protein: P00974

## 2. PDB format (4 Punkte)

In the lecture we discussed the PDB format. Make sure that you understand how a PDB file is laid out. The following section is part of a PDB file. The missing column usually denotes the amino acid that each atom belongs to. Note that the amino acid is part of a protein.

```
ATOM 231 N ??? A 15 3.802 13.375 7.749 1.00 7.12 N
ATOM 232 CA ??? A 15 2.855 14.452 7.576 1.00 9.27 C
ATOM 233 C ??? A 15 3.033 15.572 8.625 1.00 8.55 C
ATOM 234 O ??? A 15 2.791 16.739 8.344 1.00 13.11 O
ATOM 235 CB ??? A 15 1.356 13.847 7.736 1.00 11.31 C
ATOM 236 OG1 ??? A 15 1.093 12.800 6.793 1.00 14.65
O
ATOM 237 CG2 ??? A 15 0.318 14.916 7.630 1.00 13.52
C
ATOM 238 H ??? A 15 3.490 12.731 8.225 1.00 7.32 H
ATOM 239 HA ??? A 15 2.940 14.840 6.680 1.00 9.08 H
ATOM 240 HB ??? A 15 1.280 13.471 8.627 1.00 11.49 H
ATOM 241 HG1 ??? A 15 1.303 12.101 7.079 1.00 14.74
H
ATOM 242 HG21 ??? A 15 0.606 15.595 7.012 1.00 13.35
H
ATOM 243 HG22 ??? A 15 0.170 15.319 8.492 1.00 13.34
H
ATOM 244 HG23 ??? A 15 -0.510 14.540 7.317 1.00 13.1
6 H
```

Complete the missing column (PDB uses 3 letter code in upper case letters): THR

**Draw the amino acid on a sheet of paper** and label each atom with its PDB atom name.

The hydrogen with number 238 is attached to which atom (number): 231

The hydrogen with number 241 is attached to which atom (number): 236

## 3. Chou-Fasman (6,5 Punkte)

The Chou-Fasman algorithm is a simple method for the prediction of secondary structure elements in protein sequences based only on amino acid propensities. In this task we will apply it to predict alpha helices in the protein subsequence: **PGHAEIKVM**

**Step 1:** Enter  $P_{\alpha}$  (two digits after the decimal point), **class** (H, h, I, i, b, B → upper- and lowercase matters), and weight  $w_i$  (one digit after the decimal point) for the given protein sequence:

|              | P    | G    | H    | A    | E    | I    | K    | V    | M    |
|--------------|------|------|------|------|------|------|------|------|------|
| $P_{\alpha}$ | 0.59 | 0.53 | 1.24 | 1.45 | 1.53 | 1.00 | 1.07 | 1.14 | 1.20 |
| Class        | B    | B    | h    | H    | H    | I    | I    | h    | h    |
| $w_i$        | -1.0 | -1.0 | 1.0  | 1.0  | 1.0  | 0.5  | 0.5  | 1.0  | 1.0  |

**Step 2:** To find helix nuclei, form all sliding windows of length 6 and calculate their corresponding  $w_i$  sums:

| Window | Amino Acid Sequence | $w_i$ sum |
|--------|---------------------|-----------|
| 1      | PGHAEI              | 1.5       |
| 2      | GHAEIK              | 3         |
| 3      | HAEIKV              | 5         |
| 4      | AEIKVM              | 5         |

**Step 3:** Extend the first helix nucleus you find to the left and to the right (round to two digits after the decimal point)

|                            | Amino Acid Sequence | $P_{\alpha}$ sums: |
|----------------------------|---------------------|--------------------|
| 1st extension to the left  | GHAE                | 4.75               |
| 2nd extension to the left  | PGHA                | 3.81               |
| 1st extension to the right | IKVM                | 4.41               |

Chou-Fasman predicts that the following subsequence forms an alpha helix: GHAEIKVM

## Assignment 11: Threading/Homology Modeling

### 1. Fold class (1 Punkt)

Use **SCOPE** to find the fold of human alpha-lactalbumin: Lysozyme-like

Use **CATH** to find the architecture of hevein: 1-Layer Sandwich

### 2. Torsion angles (2 Punkte)

Select **all** answers that are correct:

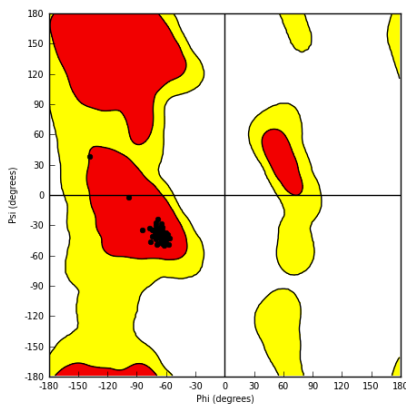
- ☐ Threading is possible even if a BLAST search against Uniprot yields no results
- ☒ The backbone torsion angles  $\phi$  and  $\psi$  have characteristic values for  $\alpha$  helices and  $\beta$  sheets
- ☒ Rotamers are stable conformations of the side chain
- ☐ The torsion angle  $\omega$  is usually fixed at 90 degrees

### 3. Threading (1,5 Punkte)

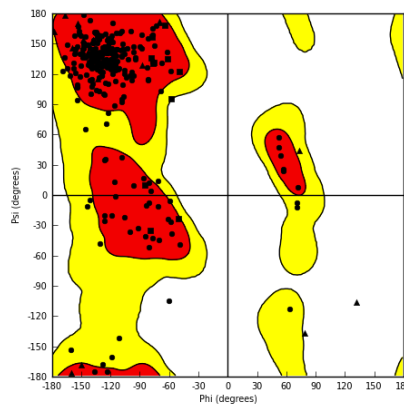
- ☒ Threading produces a backbone that can be used for homology modeling
- ☐ Threading should identify the alignment of maximal energy
- ☐ Threading produces a local sequence-structure alignment
- ☒ Threading yields the Ca positions for the given sequence
- ☒ Threading needs an energy function to solve the optimisation problem

### 4. Ramachandran Plot (3,75 Punkte)

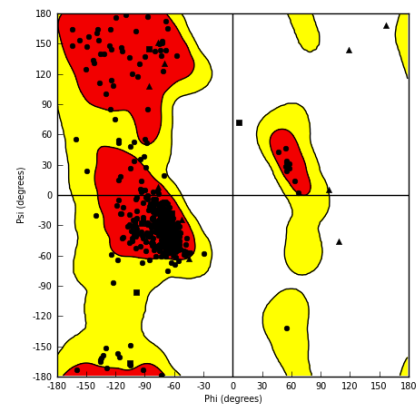
**Ramachandran plots** are a way to visualize the distribution of torsion angles  $\psi$ ,  $\phi$  of a protein structure. Make sure that you understand where these torsion angles are located and what you can learn from a Ramachandran plot. Associate the following Ramachandran plots with the corresponding PDB structure (click on the pictures to enlarge):



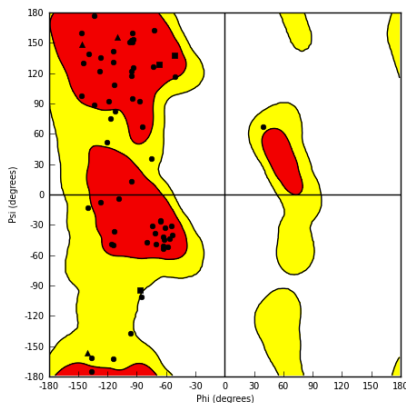
passt zu 1AIK



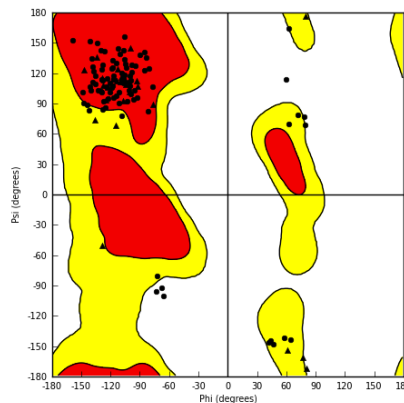
passt zu 4Y25



passt zu 5E1J



passt zu 1OWT



passt zu 2BEG

### 5. PyMOL (3,75 Punkte)

In this task we will look at the structure of the proteolytic enzyme **chymotrypsin** and its interaction with the inhibitor. A suitable structure that contains the bound inhibitor has been deposited under 1AFQ in PDB.

1. Download the structure **1AFQ** from PDB
2. Start **pymol** in the command line
3. Click on **File->Open** and select 1afq.pdb.

Investigate the structure in the viewer and use your mouse to navigate and zoom. Click on the structure in the right menu and select show **cartoon** and color by **secondary structure** (ss) and **protein chain**.

How many **chains** does the protein have? 3

How many **alpha helices** does the protein have? 3

How many **beta barrels** does the protein have (**\*NOT\*** beta sheets)? 2

Now use the command line interface and type in:

```
color grey, all
set transparency, 0.25
select inhi, resn 0fg
show sticks, inhi
```

How many **(organic) inhibitor molecules** does the PDB structure have? 2

We now want to look at the inhibitor and its **interaction with the binding pocket**. Type the following commands in the command line interface:

```
select hydrophobes,(resn ala+gly+val+ile+leu+phe+met)
select pocket, br. all within 5 of inhi
show sticks, (pocket and hydrophobes)
color orange, hydrophobes
color white, (inhi and elem c)
color red, (inhi and elem o)
color blue, (inhi and elem n)
color green, (inhi and name s)
```

Fill in the gap in the commands to answer the following question. How many **hydrophobic residues** are within **5 Å** to the inhibitor molecule of chain C? 5

How many **oxygen atoms** does the ligand have? 2

Finally we want to investigate the shape of the binding pocket:

```
show cartoon, all
show surface, pocket
```

Is the inhibitor of chain C buried in a **(concave) binding pocket**? Yes

You can save and export your visualization if you want:

```
png 1afq.png, dpi 500
```

**Optional:** The power of PyMOL lies in its command line interface and Python accessibility. In particular, you can also write Python scripts for PyMOL. For more details see: <http://www.protein.osaka-u.ac.jp/rcsfp/supracryst/suzuki/jpxtal/Katsutani/en/hydrophobicity.php>

Try to color your protein by hydrophobicity!