Introduction to Python - Python I

1 Assignmet - Python

Load the module for Python 3 with the command `module load python3/3.6.0`.
Open the Python-interpreter with the command `python3`.
You should then see at the beginning of the line: `>>>`. In this exercise we use only the Python-interpreter.
You can leave the Python-interpreter when you type `quit()`.

1. Type in the Python-interpreter the following command:

   ```
   print("Assignment7")
   ```

   What happens?

   ```
   >>> print("Assignment7")
   Assignment7
   ```

2. Enter now `i = 10` in the Python-interpreter
   and then (in a new line) `print(i)`.
   After that (in a new line) enter `j = i/2`
   and (in a new line) `print(j)`.

   Which values are displayed and why?

   ```
   >>> i = 10
   >>> print(i)
   10
   >>> j = i/2
   >>> print(j)
   5.0
   ```

   Hint: With `type()` the type of a variable can be determined. For example, `type("hello")` returns `<class 'str'>` which means that `"hello"` is of type string.

3. Assign to variable `7Assignment` the string `black magic`. Don't forget to put the string in quotation marks (`" "`).
   Which error occurs and why?

```
>>> 7Assignment = "black magic"
  File "<stdin>", line 1
    7Assignment = "black magic"
               ^
SyntaxError: invalid syntax
```

4. Assign to variable `A` a sequence `AGCTA` (don't forget to put the sequence in quotation marks). Use the built-in function `len()` to determine the length of the sequence `A` and assign the length of `A` to variable `i`. Print `A` and `i`.

```
>>> A = "AGCTA"
>>> i = len(A)
>>> print(A, i)
AGCTA 5
```

5. Concatenate `A` and `i` and print the result.
   What happens and why?

```
>>> print(A + i)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
```

6. Enter now `print(A + str(i))`.
   What happens now and why?

```
>>> print(A + str(i))
AGCTA5
```

   Hint: What might the built-in function `str()` do? There are also other built-in functions, e.g., to convert a string or number to an integer: `int()`, or to convert a string or number to a floating point: `float()`.

7. Print the substring of `A` from position 2 to 4.
   The output should be: `GCT`.

```
>>> print(A[1:4])
'GCT'
```

8. Print the prefix (beginning of a string) of length 2 and the suffix (end of a string) of length 2 of the sequence stored in `A`.
   The output should be `AG` and `TA`.

```
>>> print(A[:2])
'AG'
>>> print(A[-2:])
'TA'
```

9. Write a for-loop with the loop variable `i`, which runs from `0` to `len(A)` and prints out `i`.

   Hint: Don't forget to indent the body of the for-loop.

```
>>> for i in range(len(A)):
>>>     print(i)
```

Execute the same for-loop a second time and print out the character at each position of string A using `A[i]` as well.

```
>>> for i in range(len(A)):
>>>     print(i, A[i])
```

10. Add now an if-condition inside the for-loop, which checks if `i < len(A)/2`. Only print `i` and `A[i]` if this condition is true.

```
>>> for i in range(len(A)):
>>>     if (i < len(A)/2):
>>>         print(i, A[i])
```

11. Write a while-loop, which produces the same output as the for-loop and if-condition together.

```
>>> i = 0
>>> while (i < len(A)/2):
>>>     print(i, A[i])
```

12. Print the variable `A` again. What happens?
13. Leave the interactive mode of Python with `quit()`.
14. Now return to the interactive mode of Python and print the variable `A`. What happens now and why?

2 First small program

Open your favorite editor (nano, gedit, etc.) and write in the file named `compare.py` your first Python program.

Hint: When you type
`$ gedit compare.py&`
in the terminal, a new line in the terminal should appear
(if not press `<ctrl C>`). Then you can run your program in the same terminal window:
`$ python3 compare.py`
The advantage is that you can edit your program and switch easily between the editor and terminal window.

1. Write a short program which compares two variables `i` and `j`. It should print the value 1, if `i` and `j` are equal, and otherwise the value `0`.
2. Within the program assign different numbers to `i` and `j`, e.g.:

   a) `i = 3` and `j = 4` and
   b) `i = 10` and `j = 10`

   Does your program work?

```
2a)
i = 3
j = 4
if (i == j):
    print(1)
else:
    print(0)

2b)
i = 3
j = 4
if (i == j):
    print(1)
else:
    print(0)
```

Bonus Assignments I

1 Bonus Assignment - Sequences

In this exercise we write a short Python program (named `<program_name>.py`, think of a reasonable program name and name your file accordingly. Replace `<program_name>` with your new program name).

Choose two variables, e.g. `A` and `B` and assign the sequences `GATTACA` and `TACCATAC` to these variables. Make sure that the two sequences are assigned as strings to their variables `A` and `B`. Then print these sequences.
Save everything you wrote and close the editor. Then you can run your program: `python3 <program_name>.py`

```
A = "GATTACA"
B = "TACCATAC"
print("sequence A: ", A)
print("sequence B: ", B)
```

Then extend your program:

1. Concatenate both sequences in both ways (`AB` and `BA`) and print both options.

   ```
   A = "GATTACA"
   B = "TACCATAC"
   print("sequence A + B: ", A + B)
   print("sequence B + A: ", B + A)
   ```

2. Print prefixes and suffixes of length 3 of both sequences `A` and `B`. Use the built-in function `len()` for determining the suffixes.

   ```
   print("prefix A: ", A[:3])
   print("prefix B: ", B[:3])
   suffix_A = len(A) - 3
   suffix_B = len(B) - 3
   print("suffix A: ", A[suffix_A:])
   print("suffix B: ", B[suffix_B:])
   ```

```
# It is also possible to use a negative index
# to count from the end:
print("suffix A: ", A[-3:])
print("suffix B: ", B[-3:])
```

3. Print out the second sequence from the last to the first position (last position first, first position last).

```
for i in range(len(B)):
    print(B[len(B) - i - 1])
```

4. Assign this inverted sequence to a third variable, you could use the variable name `C`, and print the value of this variable.

```
C = ""
for i in range(len(B)):
    C = C + B[len(B) - i - 1]
print("inverted sequence B: ", C)
```

```
# One way to reverse a string is to use a negative counter
B = "TACCATAC"
C = B[::-1]
print("inverted sequence B: ", C)
```

5. Print out the middle base of each sequence. When a sequence has an even number of bases, print out the base at the right position of the middle. Use the built-in function `len()` for this task.
   For example: For `A = "GTCA"` the program should print out `C`.
   Hint: There exist built-in functions to convert a number to an integer.

```
print("middle base of A: ", A[int(len(A)/2)])
print("middle base of B: ", A[len(B)/2])
print("middle base of C: ", A[len(C)/2])
```

6. Count how often each base occurs in the first sequence (How often does `G` occur in the first sequence, then `A`, so on.) and print out this number for each base.

```
A = "GATTACA"
for i in ['A','C','G','T']:
    n = 0
    for j in A:
        if(i == j):
            n += 1
    print(i, "=", n)
```

7. Count how often `TA` occurs in the second sequence and print out this number.

```
B = "TACCATAC"
n = 0
for i in range(len(B)):
    if ((B[i] == "T") and (B[i+1] == "A")):
        n += 1
print("TA occurs ", n, " times.")
```

2 Bonus Assignment - Calculate the product of two numbers

Write in an editor the program `product.py` as introduced in the lecture, which calculates the product of two numbers `456` and `15`. Save the program code as `product.py` and run the program as described in the previous assignment.

1. Now calculate the product `234` and `24` additionally to the first product and print out both products (results) in one single line.

```
x = 456
print("x = ", x)
y = 15
print("y = ", y)
product1 = x*y

x = 234
print("x = ", x)
y = 24
print("y = ", y)
product2 = x*y

print("Products: ", product1, product2)
```

2. Change the program so that all numbers `456`, `15`, `234`, and `24` are saved in one list `l`. Change the print statement so that each number gets printed and also the product of the first two and the last two numbers.

```
l = [456, 15, 234, 24]
i = 0
while (i < 3):
    print("Product of : ", l[i], " and ", l[i+1], " is: ", l[i]*l[i+1])
    i += 2
```

3 Bonus Assignment - More sequences

Write in an editor a program, which has three lists `l`, `m`, and `n`. Each list contains several sequences. Save and run the program as described previously.

```
l: AGGTC, GATC, CTGCA, ATTCGT, ATGGT, GATC
m: CTGCA, GATC
n: CUAGCUA, GTATGG, GUAUC, GTAG
```

Note: Remember to store all sequences as strings in each of the lists.

Extend your program so that it can perform the following tasks.

1. Print each sequence in list `l`.

   ```
   l = ["AGGTC", "GATC", "CTGCA", "ATTCGT", "ATGGT", "GATC"]
   m = ["CTGCA", "GATC"]
   n = ["CUAGCUA", "GTATGG", "GUAUC", "GTAG"]
   for seq in l:
       print(seq)
   ```

2. Print the first and last sequence in list `l`.

   ```
   print(l[0], l[len(l)-1])
   ```

3. For each sequence in list `l` store the second position of each sequence in a new variable and print this new sequence.

   ```
   seq = ""
   for i in l:
       seq = seq + i[1]
   print(seq)
   ```

4. Add this new sequence to the list `l`.

   ```
   l.append(seq)
   ```

5. How long is list `l` now? Print out the length of list `l`.

```
print(len(l))
```

6. Delete the second sequence of list `l`. Print list `l` and its length.

```
l = [l[0]] + l[2:]
print(l, len(l))
```

7. Divide the new list `l` into two equal parts and store the first half in a new list `l1` and the second half in a new list `l2`. Print both lists.

```
half = len(l)/2
l1 = l[:half]
l2 = l[half:]
print("l1: ", l1)
print("l2: ", l2)
```

8. Concatenate list `l2` and list `l1` (in this order) and store it in a new list `l3`.

```
l3 = l1 + l2
print(l3)
```

9. Remove all sequences in list `l`, which are also present in list `m`.

```
for seq_m in m:
   for seq_l in l:
       if (seq_m == seq_n):
           l.remove(seq_m)
```

10. Invert all sequences in new list `l` and store them in a new list `l4`.

```
l4 = []
for seq in l:
    l4 = l4 + l[::-1]
print(l4)
```

11. In list `n` a few RNA sequences (`U` instead of `T`) are present.

    - Change these sequences back to DNA sequences.
    - Delete the RNA sequences in list `n`.
    - Add the new DNA sequences at the same position of list `n`.

    When you print list `n` it should contain the following sequences in this order: `CTAGCTA`, `GTATGG`, `GTATC`, and `GTAG`.

```
nDNA = []
for seq in n:
    if (seq.find("U") != 0):
        new_seq = seq.replace('U', 'T')
        nDNA.append(new_seq)
    else:
        nDNA.append(seq)
print(nDNA)
```

Introduction to Python - Python II

1 Assignmet - Similarity of Sequences

Write in an editor the program, which calculates the distance between two sequences `A = "ACGT"` and `B = "AGGT"`.
A simple program (without function and modules) is sufficient.

```
A = "ACGT"
B = "AGGT"

d = 0

for i in range(len(A)):
   if (A[i] != B[i]):
       d = d + 1
   else:
       d = d + 0
print("Distance between A and B: ", d)
```

1. Calculate the distance between the following sequences and print out the result. Since the following sequences are already aligned, we can calculate the distance between them. Change your program so that it can read two aligned sequences from the command line. Test your program with the following sequences.

   a) `ACGT` and `A-GT`

   b) `AC-GT` and `AGT--`

   c) `AC-CGT` and `AGT---`

   d) `ACCGT` and `TGCCA`

   e) `GATT-ACA` and `TACCATAC`

   f) `--GA--TT--AC-A` and `TA--CC--AT--CA`

```
import sys

A = sys.argv[1]
B = sys.argv[2]
```

```
d = 0

for i in range(len(A)):
    if (A[i] != B[i]):
        d = d + 1
    else:
        d = d + 0
print("Distance between A and B: ", d)
```

2. Extend the program that the aligned sequences are printed out additionally to their distance.

```
print("Sequence A: ", A)
print("Sequence B: ", B)
print("Distance between A and B: ", d)
```

3. Extend the program that the distance between two sequences is only calculated when both sequences have the same length. Test your program with the input sequences:

   a) `ACGT` and `AGT`

   b) `ACCGT` and `TGCCA`

```
import sys

A = sys.argv[1]
B = sys.argv[2]

if (len(A) == len(B)):
    d = 0
    for i in range(len(A)):
        if (A[i] != B[i]):
            d = d + 1
        else:
            d = d + 0
    print("Sequence A: ", A)
    print("Sequence B: ", B)
    print("Distance between A and B: ", d)
```

```
else:
    print("Sequences A and B are of different length.")
```

4. Extend the program that the second sequence is inverted and assigned to a third sequence. Please, read the first and second sequence from the command line. Calculate the distances between the first and the second and between the first and the third sequence. Compare the distance between the first and the second and the first and the third sequence and print the alignment with the smaller distance. If the distances are equal, then print the alignment of the first and second sequence.

   Test your program with the following sequences:

   a) `ACGT` and `A-GT`
   b) `AC-GT` and `AGT--`
   c) `ACCGT` and `TGCCA`
   d) `GATT-ACA` and `TACCATAC`

```
import sys

A = sys.argv[1]
B = sys.argv[2]

C = ""
for i in range(len(B)):
C = C + B[len(B) - i - 1]

if (len(A) == len(B)):
    distAB = 0
    distAC = 0
    for i in range(len(A)):
        if (A[i] != B[i]):
            distAB = distAB + 1
        if (A[i] != C[i]):
            distAC = distAC + 0

    if (distAB <= distAC):
        print("Sequence A: ", A)
        print("Sequence B: ", B)
        print("Distance between A and B: ", distAB)
```

```
        else:
            print("Sequence A: ", A)
            print("Sequence C: ", C)
            print("Distance between A and C: ", distAC)

    else:
        print("Sequences A and B are of different length.")
```

Bonus Assignments II

1 Bonus Assignment - Functions

Open an editor and save your new program. In this program we will create a few functions.

1. Define the two functions `similarity` and `distance`:

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

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

Note: Purines are `A` and `G`, pyrimidines are `C` and `T`.

```
pur = ["A", "G"]
pyr = ["C", "T"]

def similarity(a, b):
    if (a == b):
        return 1
    elif (((a in pur) and (b in pur)) or ((a in pyr) and (b in pyr))):
        return 0.5
    else:
        return 0

def distance(a, b):
    if (a == b):
        return 0
    elif (((a in pur) and (b in pur)) or ((a in pyr) and (b in pyr))):
        return 0.5
    else:
        return 1
```

2. Write two functions `compare1` and `compare2`, which calculate for sequences of same length the similarity and distance, respectively.

```
pur = ["A", "G"]
pyr = ["C", "T"]

def similarity(a, b):
    if (a == b):
        return 1
    elif (((a in pur) and (b in pur)) or ((a in pyr) and (b in pyr))):
        return 0.5
    else:
        return 0

def distance(a, b):
    if (a == b):
        return 0
    elif (((a in pur) and (b in pur)) or ((a in pyr) and (b in pyr))):
        return 0.5
    else:
        return 1


def compare1(A, B):
   sim = 0.0
   for i in range(len(A)):
       sim = sim + similarity(A[i], B[i])
   return sim

def compare2(A, B):
   dist = 0.0
   for i in range(len(A)):
       dist = dist + distance(A[i], B[i])
   return dist
```

3. Calculate the similarity and distance for the following sequences. Read these sequences from the command line and print out their similarity and distance.

   a) `ACGT` and `TGCA`
   b) `ATAG` and `ACAC`
   c) `ACGC` and `ATTT`
   d) `AGTT` and `ACTT`
   e) `TCGC` and `AGAG`

```
import sys

seq1 = sys.argv[1]
seq2 = sys.argv[2]

print "Similarity: ", compare1(seq1, seq2)
print "Distance: ", compare2(seq1, seq2)
```

2 Bonus Assignment - Modules

In this exercise we will write three different programs.

1. Write two Python programs (e.g., `Sim.py` and `Dist.py`) which contain respectively the two modules `similarity` and `distance` as defined previously.

```
########################
### Sim.py
########################
pur = ["A", "G"]
pyr = ["C", "T"]

def similarity(a, b):
    if (a == b):
        return 1
    elif (((a in pur) and (b in pur)) or ((a in pyr) and (b in pyr))):
        return 0.5
    else:
        return 0
```

```
########################
### Dist.py
########################
pur = ["A", "G"]
pyr = ["C", "T"]

def distance(a, b):
    if (a == b):
        return 0
    elif (((a in pur) and (b in pur)) or ((a in pyr) and (b in pyr))):
        return 0.5
    else:
        return 1
```

2. Write a third Python program that calculates for each combination of two sequences stored in list `l` the similarity and distance using the modules defined previously.
`l = ["ATCCGGT", "GCGTTAC", "CTACTGC", "TTGCAGT", "AGTCACC"]`

```
import Sim
import Dist

def compare1(A, B):
   sim = 0.0
   for i in range(len(A)):
       sim = sim + similarity(A[i], B[i])
   return sim

def compare2(A, B):
   dist = 0.0
   for i in range(len(A)):
       dist = dist + distance(A[i], B[i])
   return dist

l = ["ATCCGGT", "GCGTTAC", "CTACTGC", "TTGCAGT", "AGTCACC"]

for i in range(len(l)):
    for j in range(i+1, len(l)):
        s = s + compare1(l[i], l[j])
        d = d + compare2(l[i], l[j])
        print(l[i], l[j], " Similarity: ", s, " Distance: ", d)
```

3. Extend your third program. Determine the alignment with the highest similarity of all sequences stored in list `l`. Write these two sequences and the alignment into a new file, called `similar_sequences.txt`.

   For example for two given sequences: `"ATC"` and `"ACC"` The alignment would be:
   ```
   ATC
   | |
   ACC
   ```

And this alignment should be written to a new output file.

Hint: A line-break in Python can be made by adding `'\n'` to the end of the line.

```
import Sim

def compare1(A, B):
   sim = 0.0
   for i in range(len(A)):
       sim = sim + similarity(A[i], B[i])
   return sim

l = ["ATCCGGT", "GCGTTAC", "CTACTGC", "TTGCAGT", "AGTCACC"]

bestSim = 0
bestA = ""
bestB = ""

for i in range(len(l)):
    for j in range(i+1, len(l)):
        s = s + compare1(l[i], l[j])
        if (s > bestSim):
            bestSim = s
            bestA = l[i]
            bestB = l[j]


matches = ""
for i in range(len(bestA)):
    if (bestA[i] == bestB[i]):
        matches = matches + "|"
    else:
        matches = matches + " "

outfile = open("similar_sequences.txt", "w")
outfile.write(bestA + "\n")
outfile.write(matches + "\n")
outfile.write(bestB + "\n")
```