Homework

Python-1

Sup'Biotech 3

Python

Pierre Parutto

November 7, 2016

Preamble

Document Property

Authors	Pierre Parutto
Version	1.0
Number of pages	14

Contact

Contact the assistant team at: supbiotech-bioinfo-bt3@googlegroups.com

Copyright

The use of this document is strictly reserved to the students from the Sup'Biotech school. This document must have been downloaded from www.intranet.supbiotech.fr, if this is not the case please contact the author(s) at the address given above.

©Assistants Sup'Biotech 2016.

Contents

1 Introduction

In this homework we will apply the basic programming skills that you have just acquired to crack some biology-related questions.

1.1 File Architecture

You must respect the following architecture for your work:

```
login_1
├── AUTHORS
└── src
    └── files.py
```

You must have a folder named with your login, this folder must contain:

1. **A text file** named *AUTHORS* containing your first and last names.
2. **A folder** named *src* that contains your code files.

1.2 Submission

- Deadline: until **Thursday October 13, 23h42**;
- Submission: **a zip file** named: *login_1.zip* to upload on the bioinformatics intranet.

A bad architecture of your submission may result in a 2 points penalty.

1.3 Cheating

Basically, **DO NOT CHEAT**. Your work will be automatically tested against cheating. If two people are detected as cheaters, they will receive the **grade 0** for the homework and the administration will be told of their attempt. All detected cheaters are treated equally, I do not care who wrote the code and who took it.

2 Example

Here is an example of how to answer an homework question.

2.1 Question

File: *toto.py*

Write the function called `my_sum(a: int, b:int) -> int` that returns the sum of `a` and `b`.

2.2 Answer

The content of the file *toto.py* is thus:

```
def my_sum(a, b):
    return a + b
```

2.3 Testing your code

Although not mandatory in this first homework, **I strongly advise you to test your code**. You can do that by calling your function with some values and checking that the answer is what you expect. For example, your file *toto.py* becomes:

```
def my_sum(a, b):
    return a + b

print(my_sum(5, 9) == 16)
print(my_sum(5, 0) == 5)
print(my_sum(5, -5) == 0)
```

And the Python output is:

```
True
True
True
```

You must remove all your tests before you submit your code. In your code files I want only function declarations and nothing else.

3 Computer-science Problems

3.1 Integer Sequence (2 point)

File: *seq.py*

Consider the following sequence defined as follows for all $n \in \mathbb{N}$:

$$u_n = \begin{cases} u_0 & \text{if } n = 0 \\ 2 \times u_{n-1} \% (u_0)^2 & \text{otherwise} \end{cases}$$

Write the function `seq1(n: int, u0: int) -> int` that returns the value of $u(n)$.

Example

```
>>> seq1(0, 5)
5
>>> seq1(1, 5)
10
>>> seq1(2, 5)
20
>>> seq1(3, 5)
15
>>> seq1(4, 2)
0
```

Correction:

```
def seq1(n, u0):
    toto = u0
```

```
    i = 1
    while i <= n:
        toto = 2*toto % u0 ** 2
        i = i + 1
    return toto
```

3.2 Number Of Solutions Of A Second Order Polynome (4 points)

File: *poly.py*

A second order polynomial equation is an equation of the form:

$$a \times x^2 + b \times x + c = 0$$

As you know from highschool the number of solutions of this equation depend of the value of its discriminant Δ defined as:

$$\Delta = b^2 - 4 \times a \times c$$

If $\Delta < 0$ the equation has **no solution**, if $\Delta = 0$ the equation has **one** solution and if $\Delta > 0$ the equation has **two** solutions.

Write the function `number_of_sol(a: float, b: float, c: float) -> int` that returns the number of solution(s) of the corresponding second order equation.

Example

```
>>> number_of_sol(1, 5, 2)
2
>>> number_of_sol(3, 2, 1)
0
>>> number_of_sol(0.5, -3, 4.5)
1
```

Correction:

```
def number_of_sol(a, b, c):
    delta = b**2 - 4 * a * c
    if delta < 0:
        return 0
    elif delta == 0:
        return 1
    else:
        return 2
```

3.3 Solution Of A Second Order Polynome (4 points)

File: *solve_pol.py*

The solution of the second order equation:

$$a \times x^2 + b \times x + c = 0$$

depends on the value of $\Delta = b^2 - 4 \times a \times c$ as follows:

- If $\Delta < 0$ the equation has no solution;
- If $\Delta = 0$ the equation has the solution $x^* = \frac{-b}{2a}$;
- If $\Delta > 0$ the equation has two solutions: $x_1 = \frac{-b-\sqrt{\Delta}}{2a}$ and $x_2 = \frac{-b+\sqrt{\Delta}}{2}$.

Write the function `solve(a: float, b: float, c: float) -> int` that returns:

- `None` if the equation has no solution;
- The unique solution x^* if the equation has only one solution;
- The solution x_2 (defined above) if the equation has two solutions.

Note: To be able to compute the square root in your code you must fetch the function `sqrt` (short hand for "square root"). To do so you must put the following line at **the beginning** of your file (not in the body of the function !):

```
from math import sqrt
```

Example

```
>>> solve(1, 5, 4)
-1
>>> solve(3, 2, 1)
None
>>> solve(0.5, -3, 4.5)
3
```

Correction:

```
from math import sqrt
def solve(a, b, c):
    delta = b * b - 4 * a * c

    if delta < 0:
        return None
    elif delta == 0:
        return - b / 2 / a
    else:
        return (-b + sqrt(delta)) / 2 / a
```

3.4 Division

File: *division.py*

The result of the integer division between two two numbers $a, b > 0$ can be obtained using the following simple procedure:

- while $a \geq 0$ subtract b from a.

The number of times q that a was subtracted from b is called the quotient. The last value of a is called the remainder (we will not use it here).

Write the function `my_division(a: float, b: float) -> int` that computes the quotient (q) in the integer division of a by b as defined above.

Warning YOU **can only use** the - operator in this function. (No / nor //).

Example

```
>>> my_division(36, 6)
6
>>> my_division(37, 6)
6
>>> my_division(15, 2)
7
```

Correction:

```
from math import sqrt
def my_division(a, b):
    q = 0
    while a - b >= 0:
        a = a - b
        q = q + 1
    return q
```

4 Biology-related Problems

4.1 Translation

File: *codon.py*

A codon is a triplet of nucleotide. Each codon is associated to a specific amino acid during the translation of a mRNA sequence. The following table associates to each codon its amino acid:

1^{st}	2^{nd}	3^{rd}	AA
A	A	A	Lys
A	A	U	Asn
A	A	C	Asn
A	A	G	Lys
A	U	A	Ile
A	U	U	Ile
A	U	C	Ile
A	U	G	Met
A	C	A	Thr
A	C	U	Thr
A	C	C	Thr
A	C	G	Thr
A	G	A	Arg
A	G	U	Ser
A	G	C	Ser
A	G	G	Arg
U	A	A	STOP
U	A	U	Tyr
U	A	C	Tyr
U	A	G	STOP
U	U	A	Leu
U	U	U	Phe

1^{st}	2^{nd}	3^{rd}	AA
U	U	C	Phe
U	U	G	Leu
U	C	A	Ser
U	C	U	Ser
U	C	C	Ser
U	C	G	Ser
U	G	A	STOP
U	G	U	Cys
U	G	C	Cys
U	G	G	Trp
C	A	A	Gln
C	A	U	His
C	A	C	His
C	A	G	Gln
C	U	A	Leu
C	U	U	Leu
C	U	C	Leu
C	U	G	Leu
C	C	A	Pro
C	C	U	Pro
C	C	C	Pro
C	C	G	Pro

1^{st}	2^{nd}	3^{rd}	AA
C	G	A	Arg
C	G	U	Arg
C	G	C	Arg
C	G	G	Arg
G	A	A	Glu
G	A	U	Asp
G	A	C	Asp
G	A	G	Glu
G	U	A	Val
G	U	U	Val
G	U	C	Val
G	U	G	Val
G	C	A	Ala
G	C	U	Ala
G	C	C	Ala
G	C	G	Ala
G	G	A	Gly
G	G	U	Gly
G	G	C	Gly
G	G	G	Gly

Write the function `codon_to_AA(nuc1: str, nuc2: str, nuc3: str) -> str` that returns the abreviated name of the amino acid associated to the codon `nuc1nuc2nuc3`.

Warning: If any of the characters provided is not `AUCG` your function must return `None`.

Warning: You can only use what we have seen up to course 3 to solve this function. Advanced structures like dictionaries are **PROHIBITED**.

Note: There is an "intelligent" way to do this function.

Exemple

```
>>> codon_to_AA("A", "U", "C")
"Ile"
>>> codon_to_AA("A", "U", "E")
None
>>> codon_to_AA("G", "G", "U")
"Gly"
>>> codon_to_AA("U", "A", "A")
"STOP"
```

Correction:

```
def codon_to_AA(nuc1, nuc2, nuc3):
    if nuc1 == "A":
        if nuc2 == "A":
            if nuc3 == "A":
                return "Lys"
            elif nuc3 == "U":
```

```
            return "Asn"
        elif nuc3 == "C":
            return "Asn"
        elif nuc3 == "G":
            return "Lys"
        else:
            return None
    elif nuc2 == "U":
        if nuc3 == "A":
            return "Ile"
        elif nuc3 == "U":
            return "Ile"
        elif nuc3 == "C":
            return "Ile"
        elif nuc3 == "G":
            return "Met"
        else:
            return None
    elif nuc2 == "C":
        if nuc3 == "A":
            return "Thr"
        elif nuc3 == "U":
            return "Thr"
        elif nuc3 == "C":
            return "Thr"
        elif nuc3 == "G":
            return "Thr"
        else:
            return None
    elif nuc2 == "G":
        if nuc3 == "A":
            return "Arg"
        elif nuc3 == "U":
            return "Ser"
        elif nuc3 == "C":
            return "Ser"
        elif nuc3 == "G":
            return "Arg"
        else:
            return None
    else:
        return None
elif nuc1 == "U":
    if nuc2 == "A":
        if nuc3 == "A":
            return "STOP"
        elif nuc3 == "U":
            return "Tyr"
        elif nuc3 == "C":
            return "Tyr"
        elif nuc3 == "G":
            return "STOP"
```

```
            else:
                return None
        elif nuc2 == "U":
            if nuc3 == "A":
                return "Leu"
            elif nuc3 == "U":
                return "Phe"
            elif nuc3 == "C":
                return "Phe"
            elif nuc3 == "G":
                return "Leu"
            else:
                return None
        elif nuc2 == "C":
            if nuc3 == "A":
                return "Ser"
            elif nuc3 == "U":
                return "Ser"
            elif nuc3 == "C":
                return "Ser"
            elif nuc3 == "G":
                return "Ser"
            else:
                return None
        elif nuc2 == "G":
            if nuc3 == "A":
                return "STOP"
            elif nuc3 == "U":
                return "Cys"
            elif nuc3 == "C":
                return "Cys"
            elif nuc3 == "G":
                return "Trp"
            else:
                return None
        else:
            return None
    elif nuc1 == "C":
        if nuc2 == "A":
            if nuc3 == "A":
                return "Gln"
            elif nuc3 == "U":
                return "His"
            elif nuc3 == "C":
                return "His"
            elif nuc3 == "G":
                return "Gln"
            else:
                return None
        elif nuc2 == "U":
            if nuc3 == "A":
                return "Leu"
```

```
            elif nuc3 == "U":
                return "Leu"
            elif nuc3 == "C":
                return "Leu"
            elif nuc3 == "G":
                return "Leu"
            else:
                return None
        elif nuc2 == "C":
            if nuc3 == "A":
                return "Pro"
            elif nuc3 == "U":
                return "Pro"
            elif nuc3 == "C":
                return "Pro"
            elif nuc3 == "G":
                return "Pro"
            else:
                return None
        elif nuc2 == "G":
            if nuc3 == "A":
                return "Arg"
            elif nuc3 == "U":
                return "Arg"
            elif nuc3 == "C":
                return "Arg"
            elif nuc3 == "G":
                return "Arg"
            else:
                return None
        else:
            return None
    elif nuc1 == "G":
        if nuc2 == "A":
            if nuc3 == "A":
                return "Glu"
            elif nuc3 == "U":
                return "Asp"
            elif nuc3 == "C":
                return "Asp"
            elif nuc3 == "G":
                return "Glu"
            else:
                return None
        elif nuc2 == "U":
            if nuc3 == "A":
                return "Val"
            elif nuc3 == "U":
                return "Val"
            elif nuc3 == "C":
                return "Val"
            elif nuc3 == "G":
```

```
            return "Val"
        else:
            return None
    elif nuc2 == "C":
        if nuc3 == "A":
            return "Ala"
        elif nuc3 == "U":
            return "Ala"
        elif nuc3 == "C":
            return "Ala"
        elif nuc3 == "G":
            return "Ala"
        else:
            return None
    elif nuc2 == "G":
        if nuc3 == "A":
            return "Gly"
        elif nuc3 == "U":
            return "Gly"
        elif nuc3 == "C":
            return "Gly"
        elif nuc3 == "G":
            return "Gly"
        else:
            return None
    else:
        return None
else:
    return None
```

4.2 Simulating The Noyes-Whitney Equation

File: *noyes_whitney.py*

The Noyes-Whitney equation describes the rate of dissolution in mass unit / time (for example kg/s) of a solid in a liquid . It is very used in pharmacology where the solid is a pill and the liquid can be water or blood or any physiological fluid.

The equation is stated as follows:

$$\frac{dW(t)}{dt} = \frac{D \times A \times (C_s - \frac{W(t)}{V})}{L}$$

where:

- $t \geq 0$ is the time in time units (for example s);
- $W(t)$ is the quantity of dissolved solid molecules at time t in mass units (for example kg);
- D is the diffusion coefficient of the solid molecule in the liquid in surface units squared / time (for example $\mu m^2/s$);
- A is the surface area of the solid in surface units (for example μm^2);

- C_s is the saturation concentration of the solid molecule in the solution in mass / volume units (for example mg/L);
- V is the volume of the solution in volume units (for example ml). Note that $\frac{W(t)}{V}$ is the concentration of the solid molecule in the solution.
- L is the thickness of the diffusion layer in distance units (for example μm).

Write the function:

`simu_nw(W0: float, A: float, D: float, V: float, Cs: float, L: float, dt: float t: int) -> float`

that returns the quantity $W(t)$ given all the parameters of the equation `A,D,C,Cs,L`, the initial mass of dissolved solid molecule `W0`, the timestep for the simulation `dt` and the time to stop `t`.

Note: To compute $W(t)$ from the equation, you will use the Euler approximation (see also exercise from Lab1) stated as follows:

Considering a function f depending of one parameter t and its derivative $\frac{df}{dt}$, the Euler scheme is:

$$f(t+dt) = f(t) + \frac{df(t)}{dt} \times dt$$

where dt is called the simulation timestep, the smallest it is the more precise the result will be.

To get the value of f at some time τ, you have to compute all values from the time $t = 0$, for which the value $f_0 = f(t_0)$ is known, until you reach $t = \tau$ using the Euler scheme. At each iteration the time increases by dt.

Example

```
>>> A = 2800
>>> Cs = 20
>>> L = 0.05
>>> D = 1.6e-6
>>> W0 = 0
>>> V = 5.1
>>> simu_nw(W0, A, D, V, Cs, L, 0.0001, 10)
16.43414
>>> simu_nw(W0, A, D, V, Cs, L, 0.0001, 30)
41.78548
>>> simu_nw(W0, A, D, V, Cs, L, 0.0001, 60)
66.45305
>>> A = 50000
>>> simu_nw(W0, A, D, V, Cs, L, 0.0001, 10)
97.57324
>>> simu_nw(W0, A, D, V, Cs, L, 1, 10)
99.63637
```

The Figure 1 presents results of the simulation with the previous parameters.

Correction:

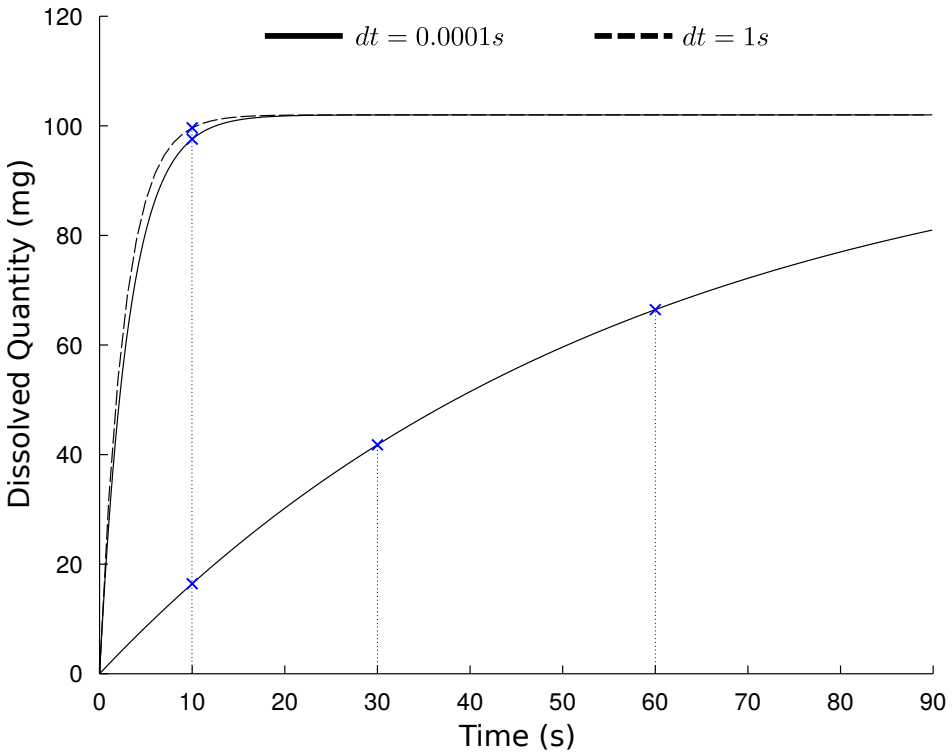

Figure 1: Simulation of the Noyes-Withney equation. Top lines corresponds to $A = 50000$ and the bottom solid line to $A = 2800$.

```
def simu_nw(W0, A, D, V, Cs, L, dt, t):
    cur_t = 0
    w = W0
    while cur_t < t:
        dw_dt = (A * D * (Cs - w/V)) / L
        w = w + dw_dt * dt
        cur_t = cur_t + dt
    return w
```