Final
Bioinformatics

Sup'Biotech 3
Python

Pierre Parutto

August 28, 2016

Nom:	
Prnom:	
Groupe:	

Documents autoriss: No document; No calculator.

Rpondre sur le sujet dans les cadres prvus cet effet. Si vous manquez de place, continuez au verso de la page.

1 Cours (8 points)

1.1 QCM (5 points)

Multiple answers can be correct; For each question: all good answers: **+0.5 point**; a bad answer: **-0.25 point**; No answer or missing good answers: **0 point**. If the total grade of the mcq is < 0 the grade is set to 0.

Questions	Answers
1. If a, b, q, r are four integers such that `a = b*q + r` then:	■ `q = a//b` ■ `r = a%b` □ `q = a / b` □ `r = a * b`
2. `5 * 8 % 2 < 4 * 5 + 3`	■ `True` □ `5` □ an Error □ `4`
3. If I import the variable `a` from the file `toto.py` with the command `import toto`, how do I access to `a` ?	□ `a` □ I cannot it is lost □ `a.a` ■ `toto.a`
4. The code `"123"+3` produces:	□ `"456"` □ `"1233"` ■ an error □ `"3123"`
5. About the `while` construction:	■ it is possible to use a `while` construction inside a `while` construction. ■ it is possible that it does not terminate □ the values in the condition must **never** change □ it is **mandatory** to use a counter variable
6. `range(1,11,2)` produces :	■ `[1,3,5,7,9]` □ `[1,2,3,4,5,6,7,8,9,10]` □ `[0,2,4,6,8,10]` □ `[1,3,5,7,9,11]`

7. After executing the following code, the value of the variable `l` is ?
`inl = [1,2,"a",True,None]`
`l = []`
`for e in inl:`
   `l.append(e)`
	□ `None`
	□ an error
	■ `[1,2,"a",True,None]`
	□ `[0,1,2,3,4]`

	Continued on the next page...

Questions	Answers
8. A recursive function on a list `l` generally:	■ treats the first element `l[0]` and calls itself on the remaining list `l[1:]` ■ has the empty list `[]` as a base case ☐ do not work on empty lists ☐ use a `for` loop to go through the list
9. What proposition(s) create(s) a 3 × 3 integer matrix (of type `array`) containing only the value 2?	■ `2 * ones((3,3))` ■ `ones((3,3)) + ones((3,3))` ☐ `twos((3,3))` ■ `array([[2,2,2], [2,2,2], [2,2,2]])`
10. In a matrix:	■ the first index represents the lines, the second represents the columns ☐ the first index represents the columns, the second represents the lines ☐ all dimensions **must** have the same size ☐ a matrix cannot have more than two dimensions

1.2 Questions (3 points)

1.3 Code Blocs (1 point)

Represent **all** the blocs of codes by vertical lines at the left of the following code:

```
def my_function(a,n):
    res = 0
    i = 0
    while i < n:
        j = 0
        while j < n:
            if i * j % 3 == 2:
                res = res + i + j
            elif i * j % 5 == 0:
                res = res + i * j
            j = j + 1
        i = i + 1
    if res % 2 == 1:
        res = res + 1
    return res

a = 2
my_function(a, 5)
```

Correction:

```
def my_function(a, n):
    res = 0
    i = 0
    while i < n:
        j = 0
        while j < n:
            if i * j % 3 == 2:
                res = res + i + j
            elif i * j % 5 == 0:
                res = res + i * j
            j = j + 1
        i = i + 1
    if res % 2 == 1:
        res = res + 1
    return res

a = 2
my_function(a, 5)
```

1.4 Evaluation (1 point)

Give the result of each of the following expressions (0.25 point each):

```
>>> 5 + 3 * 2
>>> 5 + 3 * 2 ** 2
>>> 5 % 3 * 2 + 5
>>> (4 * 6) + (4 % 3) % 2
```

Correction:

```
11
17
9
25
```

1.5 Recursive Calls (1 point)

Consider the following recursive function:

```
def secret(n):
    if n == 0 or n == 1:
        return 1
    if n % 2 == 0:
        return n * secret(n-1)
    return secret(n-1) + secret(n-2)
```

Write the list of all the recursive calls done when evaluating `secret(5)` and give the result of this evaluation.

Correction:

```
              15
  secret(5)   =   secret(4)  +  secret(3)
                     ‖ 12          ‖ 3
                 4*secret(3)    secret(2)  +  secret(1)
                     ‖ 3           ‖ 2           ‖
 secret(2) + secret(1)       2*secret(1)         1
      ‖ 2        ‖               ‖
  2*secret(1)    1               1
      ‖
      1
```

2 Python (7 points)

1 point for each correct function.

2.1 Sequence

Consider the following sequence u_n:

$$\begin{cases} u_{n+1} & = & 2 * u_n + (u_n)^2 \\ u_0 & = & u0 \end{cases}$$

Write a **recursive** function `seq(n: int, u0: int) -> float` that returns the value u_n.

Example

```
>>> seq(1, 1)
3
>>> seq(2, 1)
15
>>> seq(3, 1)
255
```

Correction:

```
def seq(n, u0):
    if n == 0:
        return u0
    return 2 * seq(n-1, u0) + seq(n-1, u0) ** 2
```

2.2 DNA Or RNA?

Write a function `is_dna_or_rna(s: str) -> bool` that returns `True` if the string `s` contains only the characters `A`,`T`,`G`,`C`,`U` and `False` otherwise. If `s` is empty, the function must return `True`.

Example

```
>>> is_dna_or_rna("")
True
>>> is_dna_or_rna("ATGTGTA")
True
>>> is_dna_or_rna("ATGTGTAUCA")
True
>>> is_dna_or_rna("ATGTGTAUEEECA")
False
```

Correction:

```
def is_dna_or_rna(s):
    for e in s:
        if e != "A" and e != "T" and e != "G" and e != "C" and e != "U":
            return False
    return True
```

2.3 Sum in List

Given a list of integers `l` of length N we want to compute the following sum:

$$\sum_{i=0}^{N} \sum_{j=0}^{l[i]} j$$

The sum for each element `e` of the list `l` of the sum of the integers from 0 to `e` **included**.

Write a function `sum_list(l: list) -> int` that returns the sum described above for the list of integers `l`.

Example

```
>>> sum_list([1,2])
4
>>> sum_list([1,2,3])
10
>>> sum_list([0,4,5,1])
26
```

Correction:

```
def sum_list(l):
    res = 0
    for e in l:
        for i in range(e+1):
            res = res + i
    return res
```

2.4 Read and Count

Write a function `count_A(fname: str) -> int` that returns the number of adenine in all the DNA sequences stored in the file named `fname`. The file `fname` contains different sequences separated by `"\n"` characters.

Example

With the three files:

- *seq1.txt*:

```
GTGTGCG
GCGCGGC
```

- *seq2.txt*:

```
ATTGAT
AGGCA
ACCACA
```

- *seq3.txt*:

```
AAAATA
GCGCG
AAAAA
```

```
>>> count_A("seq1.txt")
0
>>> count_A("seq2.txt")
7
>>> count_A("seq3.txt")
10
```

Correction:

```
def count_A(fname):
    f = open(fname, "r")
    res = 0
    for line in f:
        for b in line:
            if b == "A":
                res = res + 1
    f.close()
    return res
```

2.5 Recursive Writing

Write a **recursive** function `rec_write(f: File, l: list) -> None` that writes into the file `f` (`f` is opened in writing mode) the content of the list of strings `l`. The last line of the generated file must be `"END\n"`.

Example

```
>>> f = open("1.txt", "w")
>>> rec_write(f, [])
>>> f.close()
>>> f = open("2.txt", "w")
>>> rec_write(f, ["1", "2A"])
>>> f.close()
>>> f = open("3.txt", "w")
>>> rec_write(f, ["AA", "BB", "CC"])
>>> f.close()
```

Produces the three files:

- `1.txt`:

```
END
```

- `2.txt`:

```
1
2A
END
```

- `3.txt`:

```
AA
BB
CC
END
```

Correction:

```
def rec_write(f, l):
    if l == []:
        f.write("END\n")
    else:
        f.write(l[0] + "\n")
        rec_write(f, l[1:])
```

2.6 Sum Of Matrix Elements

Write a function `mat_sum(m: array) -> int` that returns the sum of all the elements in the matrix `m`.

Example

```
>>> from numpy import array
>>> mat_sum(array([[1,2], [1,1]]))
5
>>> mat_sum(array([[1,2,3], [4,5,6]]))
21
```

```
>>> mat_sum(array([[1,2,3], [4,5,6], [7,8,9]]))
45
```

Correction:

```
def mat_sum(m):
    res = 0
    i = 0
    while i < m.shape[0]:
        j = 0
        while j < m.shape[1]:
            res = res + m[i,j]
            j = j + 1
        i = i + 1
    return res
```

2.7 Creating A Matrix

You are given two sequences corresponding to the same gene in two different organisms. We will consider only point mutation (modification of a single base at some position) and want to evaluate the number of mutations from one base into another.

Write a function `pm_count(s1: str, s2: str) -> array` that returns a 4×4 matrix containing the number of observed point mutations between the two strings `s1` and `s2`.

- The two strings `s1` and `s2` have the same size;
- Consider the following mapping: `A` $\to 0$, `T` $\to 1$, `G` $\to 2$, `C` $\to 3$;
- **Do not count** when the bases in the two sequences are the identical (leave the diagonal elements to 0).

Example

```
>>> pm_count("TATACGC", "TATACGC")
array([[0,0,0,0], [0,0,0,0], [0,0,0,0], [0,0,0,0]])
>>> pm_count("ATTGT", "AAAAC")
array([[0,0,0,0], [2,0,0,1], [1,0,0,0], [0,0,0,0]])
>>> pm_count("GCGAGC", "ACGCAT")
array([[0,0,0,1], [0,0,0,0], [2,0,0,0], [0,1,0,0]])
```

Correction:

```
from numpy import zeros
def pm_count(s1, s2):
    m = zeros((4,4))
    map = {"A":0,"T":1,"G":2,"C":3}
    for i in range(len(s1)):
        if s1[i] != s2[i]:
            m[map[s1[i]],map[s2[i]]] = m[map[s1[i]],map[s2[i]]] + 1
    return m
```

3 A Game Of Tree (5 points)

Consider the tree in Figure 1 (on page 13), it represents the lineage tree of the Targaryen family. In this tree, boxes are individuals, plain boxes represent males and dashed boxes females, an horizontal link between two people means that they are partners. When a partner is someone exterior to the family, the name of his family is also written (on the second line). When the name of an individual or its family are unknown a `?` is put.

Note about the code: In this section you are free to organize your code as you want, use as much functions and variables, as long as it provides an answer to the question.

3.1 In Python (1 point)

We want to represent this tree in Python. I defined a new type called `Tree` where the values are either `None` or a list of 5 elements:

1. `e[0]` (type: `str`): the name of the individual;
2. `e[1]` (type: `str`): the family of the individual;
3. `e[2]` (type: `str`): the sex of the individual (`"M"` for male or `"F"` for female);
4. `e[3]` (type `Tree`): the partner;
5. `e[4]` (type `list`: the list of the children.

Note About The Partner Field As we consider only the Targaryen lineage, in order to prevent cross links:

- If one partner is exterior to the Targaryen family, its `partner` value (`e[3]`) is set to `None` and its `children` value (`e[4]`) is set to the empty list.
- If the two partners are from the Targaryen Familly, the female's `partner` value (`e[3]`) is set to `None` and its `children` value (`e[4]`) is set to the empty list.

For example, here is the Python value for the tree of Figure 1 starting at the node `Baelor` (and representing its partner):

```
["Baelor", "Targaryen", "M", ["?", "?", "F", None, []], \
[["Valarr", "Targaryen", "M", None, []], ["Matarys", "Targaryen", "M", None, []]]]
```

Give the Python value for the tree of Figure 1 **starting at the node** `Aerys II` (and representing its partner).

Correction:

```
["AerysII", "Targaryen", "M", ["Rhaelle", "Targaryen", "F", None, []], \
  [["Rhaegar", "Targaryen", "M", ["Elia", "Martell", "F", None, []], \
    [["Rhaenys", "Targaryen", "M", None, []], \
     ["Aegon", "Targaryen", "M", None, []]]], \
   ["Viserys", "Targaryen", "M", None, []], \
   ["Daenerys", "Targaryen", "F", ["Khal␣Drogo", "Dothraki", "M", None, []], []]]]
```

3.2 Consanguinity (1 + 1 points)

We consider a strict definition of consanguinity: A relation is consanguineous when the two partners have the same parents (they are sibling).

Give the **detailed** principle of an algorithm that would return the list of the names of the two partners (a tuple) of all consanguineous relations in a tree of type `Tree`.

Correction:

We consider a consanguineous relation when the two partners have the same family name. We will go through the tree and for each node `t`:

1. We first check if the partner field (`t[3]`) is not `None` (no partner);
2. Otherwise we check the family name of the partner (`t[3][1]`).

We accumulate in a list the result of calling the function on the children (`t[4]`).

Give the Python implementation of this algorithm. If you use supplementary arguments and/or functions, provide the type of their arguments and return values.

Correction:

```python
def consanguineous(t):
    if t == None:
        return []
    res = []
    if t[3] != [] and t[3][1] == t[1]:
        res.append((t[0], t[3][0]))
    for tt in t[4]:
        res = res + consanguineous(tt)
    return res
```

3.3 Blood Of The Dragon (1 + 1 points)

Targaryen love to say that they are the "blood of the dragon", let's look at this fact. Consider that by blood we understand genome and that `Viserys II` (the root of the tree in Figure 1) has 100% a blood of dragon. A child gets half his blood from the mother and half from the father. An individual exterior to the familly has 0% blood of the dragon.

Give the **detailed** principle of an algorithm that would compute the percentage of blood of the dragon that is present in a successor of `Viserys II`. You are given two values: the name of the successor (of type `str`) and the tree (of type `Tree`) and must return a value of type `float`.

Correction:
We consider the root of the tree has 100% blood of the dragon.

We go through the tree, at each node, the value of the children is:

- $\frac{1}{2}$ of the value of the children if one of the parent is not a Targaryen;
- The children value if the two parents are Targaryen;
- The case where the two parents are not Targaryen is not possible.

We will use a small trick here to make sure we return the value for the desired person. When we reach the node representing this person the function will return 1. When we reach a leaf we will return 0. We will make the sum between the different path and as we multiply the value for each node, only the node returning 1 will be considered.

Give the Python implementation of this algorithm. If you use supplementary arguments and/or functions, provide the type of their arguments and return values.

Correction:

```python
def dragonBlood(t, name):
    if t == None:
        return 0
    if t[0] == name:
        return 1

    n = 1
    if t[3] != None and t[3][1] != "Targaryen":
        n = 0.5

    res = 0
    for tt in t[4]:
        res = res + dragonBlood(tt, name)
    return n * res
```

Use this box if you need more space.

Correction:

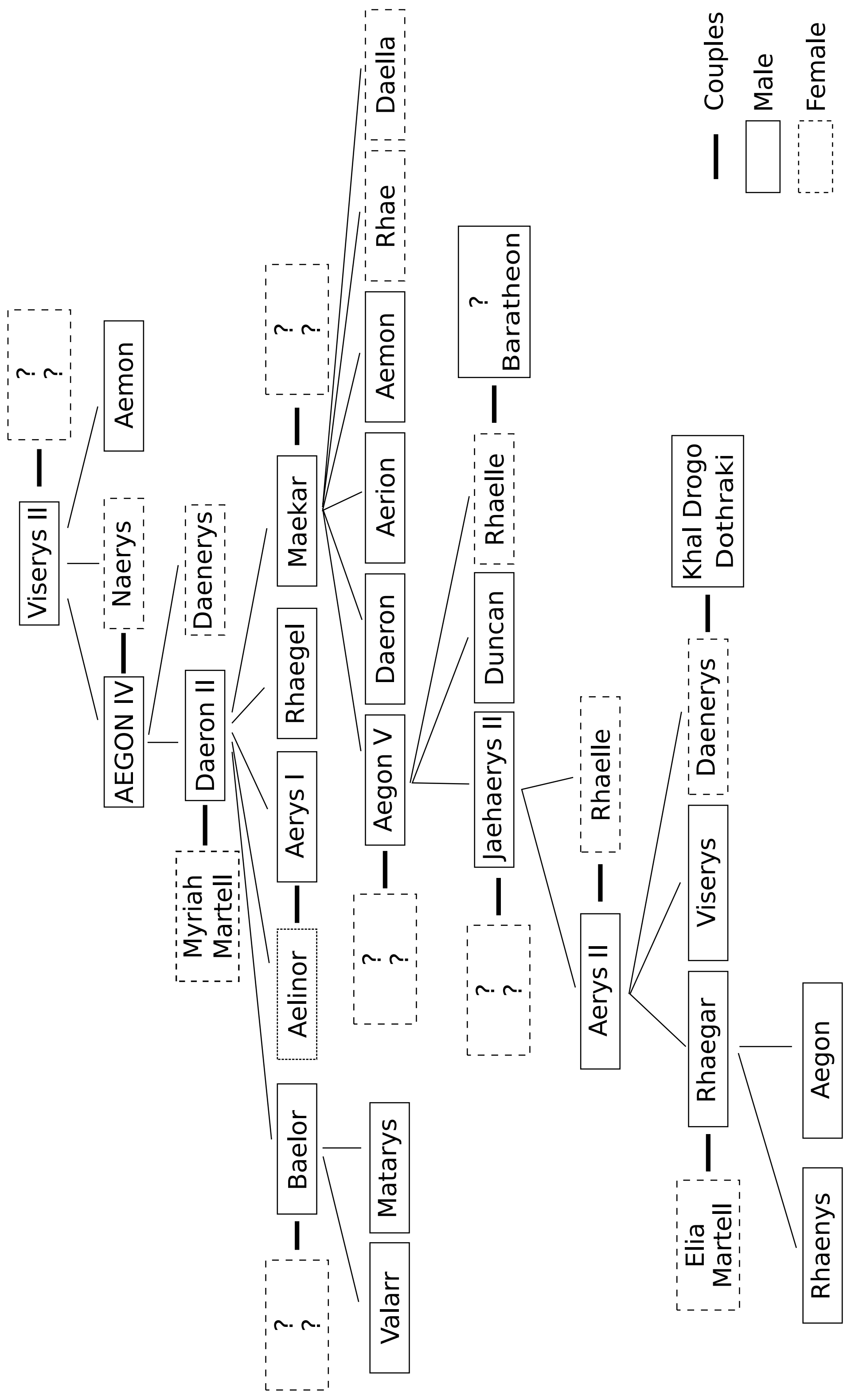

Figure 1: Targaryen Familly Tree (you can detach this page).