CS 101: Introduction to Computer Science : Solution

Mahindra École Centrale

Duration: 2 hours 30 minutes | *Final Semester Examination* | Total 100 marks

May 6th, 2015

Remark: There was two different set of question papers, the set 1 had the MCQ first (Pb I), the set 2 had the fill-in-the-blanks first (Pb I).

This solution is based on the set 1.

Problem I: MCQ (*Marks:* 30)

This first problem is a list of **Multiple Choice Question**. Each question (from Q.I.1 to Q.I.15) carries 2 marks, and has only one correct answer. You need to write on your answer paper the correct answers, and their question numbers, like: *Problem I: 1) a 2) c 3) d . . . 15) d* etc.

Q.I.1) When comparing time complexity of two algorithms, the Big O notation does not consider multiplicative and additive constants. Identify the only option that **does not** explain a reason for this:

a) these constants depend on the specific details of the way an algorithm is converted to code,

b) these constants only create confusion,

c) as n becomes large, the dependence of Big O on n significantly dominates over the effects of these constants,

d) depending on the operating system and machine architecture, these constants may impact computation times in different ways.

> **Answer:** b), *these constants only create confusion* is scientifically not rigourous enough. The three other reasons are correct, as explained in class.

Q.I.2) In many divide-and-conquer algorithms, the time complexity has a term $\log_2(n)$ because:

a) $\log_2(n)$ is a function that is very common,

b) from one step to the next, the size of the problem is reduced by half, so there are totally $\log_2(n)$ such steps,

c) it is very easy to compute the $\log_2(n)$ function,

d) $\log_2(n)$ is much smaller than n, particularly at large values of n.

> **Answer:** b), *from one step to the next, the size of the problem is reduced by half, so there are totally* $\log_2(n)$ *such steps.* We saw some examples of such behaviour, mainly with the Merge Sort algorithm.

Q.I.3) Which Python statement is used to force a specific exception?

a) `try` **b)** `except` **c)** `raise` **d)** `force`.

> **Answer:** c), *raise* is used to an exception. For example, `raise ValueError("This is a warning message.")`.

Q.I.4) The difference between a list and a tuple is that:

a) Lists are immutable while tuples are not,

b) Tuples need to contain only variables of type `str` (ie. strings),

c) Tuples are immutable while values in lists can be changed,

d) A list can be converted to a tuple, while the reverse is not possible.

> **Answer:** c), *tuples are immutable while values in lists can be changed* indeed. We explained many times that a `tuple` in Python is nothing but an immutable list.

Q.I.5) Which of the following is not a specific characteristic of **Object Oriented Programming** (OOP)?

a) Encapsulation **b)** Modularity **c)** Abstraction **d)** Inheritance.

> **Answer:** b), *modularity* is not a specific characteristic of Object Oriented Programming (OOP), but of modules/packages programming, while the others are.

Q.I.6) In the for loop below there is one elementary semantic mistake:

```
1 from math import sqrt
2 m = input("Enter a non-negative ↩
    number less than 10: ")
3 for i in xrange(0, m**2):
4     y = i + sqrt(i) + 4
5     i -= 1
6     print "i =", i, "and y =", y
```

a) `m**2` can be calculated outside the loop instead of repeating it,

b) square root of negative number is not possible,

c) the looping variable `i` should not be changed inside the for loop,

d) the print statement is incorrect.

> **Answer:** c), *the looping variable `i` should not be changed inside the for loop*, the three other issues are not happening here.

Q.I.7) In Python (and in OOP programming in general), a class is a:

a) Built-in data type, **b)** derived data-type, **c)** user-defined data type, **d)** extended data type.

> **Answer:** c), a class in Python is like a data type, defined by the user. In labs we saw examples of banking relating classes: clients were instances of the `customer` class, and accounts were instances of the `account` class.

Q.I.8) Asking Python to compute `a = 5 / '0'` will result in the exception

a) `ZeroDivisionError` **b)** `OperandError` **c)** `TypeError` **d)** `OperatorError`.

> **Answer:** c), `TypeError unsupported operand type(s) for /: "int" and "str"` will be raised, because before actually performing the operation, Python checks if he can do it, and here an integer (`5`) and a string (`"0"`) are incompatible for the `/` operator).

Q.I.9) If the file `myfile.txt` does **not** exist in the current folder, what does the statement `f=open('myfile', 'w')` will create?

a) an `IOError` exception
b) an `ImportError` exception
c) a new file object `f`, with name `'myfile.txt'`
d) a `FileError` exception.

Answer: c), because the request mode `"w"` is `write` mode, so it does not care about the fact that `myfile.txt` does not exist.

Remark: Only one student saw that there was a confusion between `"myfile"` and `"myfile.txt"`. Let me clear this out : it was simply a typing mistake. Even on Windows, specifying the text extension is always mandatory (in Python).

So if we create the file object `f` with `f = open("myfile.txt", "w")`, its name attribute will be `"myfile.txt"`.

Q.I.10) What is the output when the code below is executed?

```
1 tinylist = [2015, 'Anil']
2 print tinylist * 2
```

a) `[2015, 'Anil', 2015, 'Anil']`.
b) `[4030, 'Anil2']`,
c) `[4030, 'Anil', 'Anil']`,
d) `[20152, 'Anil2']`,

Answer: a), is the only correct answer, indeed, for a list, the right multiplication with an integer is concatenating the list with itself : `l * k = l + l + ... + l` (`k` times)).

Q.I.11) What is the value of this Python expression `3*1**3`?

a) `1` **b)** `3` **c)** `9` **d)** `27`

Answer: b), because the `**` power operator has a higher priority that the `*` multiplicative operator, so `3*1**3 = 3 * (1**3) = 3 * 1 = 3`.

Q.I.12) What is the output when the code below is executed?

```
1 d = { 'name': 'Zara', 'age': 29 }
2 print d.has_key('age'), "Name" in d, d.has_key('job')
```

a) `True, False, False` **b)** `29, False, True` **c)** `True, True, False` **d)** `29, True, False`.

Answer: a), because `d` has a key named `"age"`, but no key named `"Name"` (**everything** in Python is case sensitive!) neither a key named `"job"`).

Q.I.13) What is the output when the code below is executed?

```
1 def changeit(mylist):
2     mylist = [1, 2, 3, 4]
3     print "List inside the function:", mylist
4
5 mylist = [6, 5, 2015]
6 changeit(mylist)
7 print "List outside the function:", mylist
```

a) `List inside the function:, [6, 5, 2015]` and `List outside the function:, [1, 2, 3, 4]`.
b) `List inside the function:, [1, 2, 3, 4]` and `List outside the function:, [1, 2, 3, 4]`.

c) `List inside the function:, [1, 2, 3, 4]` and `List outside the function:, [6, 5, 2015]`.

d) `List inside the function:, [6, 5, 2015]` and `List outside the function:, [6, 5, 2015]`.

> **Answer:** c) is the only correct choice, indeed the function will not modify the list (on line 2) but create a new variable, so inside the function the list is `[1, 2, 3, 4]` but not afterward. *This was a tricky question, so both b) and c) were considered as correct.*

Q.I.14) What is printed by the code below?

```
1 a, b = 0, 3
2 while not (a == 5 or b == 5):
3     a = b
4     b = b + 1
5 print "a =", a, "and b =", b
```

a) `a = 5 and b = 5`,

b) `a = 5 and b = 6`,

c) `a = 4 and b = 5`,

d) `a = 5 and b = 4`.

> **Answer:** c), because the while loop will conserve the fact that $a \leqslant b$, and therefore stops as soon as `b = 5`.

Q.I.15) Suppose `d = { 'john': 40, 'peter': 45 }`, to delete the entry for `'john'` what should we write?

a) `del d['john']`

b) `del d('john': 40)`

c) `d.delete('john': 40)`

d) `d.del('john')`

> **Answer:** a) is the only correct syntax, as seen in class and in lab.

Problem II: Fill in the blanks (*Marks:* 20)

For this problem, you need to fill-in the blanks (`________`) for each question, on your answer booklet.
Write only the numbers of the questions and the answers for their blanks in your answer sheet, do **not** copy the entire statement.

- TODO: add some extra explanations?

Q.II.1) When you have to perform an *iterative* calculation, you would prefer to use a `________` loop when the number of cycles depends on the calculations within the iterative block, and a `________` loop when the number of iterations is independent of the block.

> **Answer:** `while`, `for`.

Q.II.2) A list can be looked upon as a special case of a `___________` where the keys are taken by default to be `___________` starting at zero in an `___________` order of appearance; so that the sanctity of their order has to be maintained as a trade-off for making keys redundant.

> **Answer:** `dict` (dictionary), integers, increasing (or ascending).

Q.II.3)

```
1  class A(objects):
2      def f(self):
3          return self.g()
4
5      def g(self):
6          return "Hi from class A."
7
8  class B(A):
9      def g(self):
10         return "Hi from class B."
```

```
1  a = A()
2  b = B()
3  print a.f(), b.f()
4  print a.g(), b.g()
```

The output for this program is _____ and _____.

Answer:

- `"Hi from class A."`, `"Hi from class B."`,
- and then `"Hi from class A."`, `"Hi from class B."`.

Q.II.4) Objects of a class are like instances of that class, where the _____ defined in the class are automatically acquired by all objects, while the _____ belongs specifically and uniquely to each object.

Answer: methods, attributes.

Q.II.5) Python exceptions can be caught using _____ and _____.

Answer: `except` and `finally` (but `try` and `except` was accepted also).

Q.II.6)

```
1  d_num = { '1': 1, '2': 2 }
2  theCopy = d_num
3  sum1 = d_num['1'] + theCopy['1']
4  d_num['1'] = 5
5  sum2 = d_num['1'] + theCopy['1']
6  print "sum1 =", sum1, "and sum2 =", sum2
```

What are the values of `sum1` = _____ and `sum2` = _____ when the below code gets executed?

Answer: `sum1 = 2` and `sum2 = 10`.

Q.II.7) A dictionary can be looked upon as a special case of a _____, where the order of appearance of values is made redundant by tagging a key to each value for its unique and explicit identification.

Answer: `list`.

Q.II.8)

```
1  x = 0
2  y = 1
3  for n in [5, 4, 6]:
4      x = x + y*n
5      y = y + 2
6  print "x =", x, "and y =", y
```

What is the value of x and y when the below code gets executed: `x =` _____ and `y =` _____ ?

Answer: `x = 47 and y = 7`. Indeed:

- $y = 1$, then 3, then 5, then **7**,
- $x = 0$, then $0 + 1 * 5 = 5$, then $5 + 3 * 4 = 17$, then $17 + 5 * 6 =$ **47**.

Q.II.9) The exact line of code to open a file, stored in the current folder, and named `"finalexam.txt"`, in order to be able to write to that file with the object `outfile`, is `_____`.
To read the first five characters from a file object called `infile`, we should use `_____`.

Answer:

- `outfile = open("finalexam.txt", "w")`.
  "Write and read" modes such that `"rw"` or `"wr"` or `"aw"` or `"wa"` were also accepted.
- `infile.read(5)` reads exactly 5 characters from the file object `infile`.

(Note: of course, single quotes are also accepted.)

Q.II.10)

```
1 d_num = {}
2 d_num[(1, 2, 4)] = 8
3 d_num[(4, 2, 1)] = 10
4 d_num[(1, 2)]    = 12
5
6 sum1 = 0
7 for k in d_num:
8     sum1 += d_num[k]
```

After having executed that code, what are the values of `len(d_num) = _____` and `sum1 = _____` ?

Answer: `len(d_num) = 3` and `sum1 = 30`, it is quite simple to get, what is done in this for loop is simply a sum of all the values `d_num[k]`, ie. $8 + 10 + 12 = 30$.

Q.II.11) What is the *usual and recommended way* to import the `numpy` and `pyplot` packages for doing scientific computations and plotting?

```
1 import ___________________  # for numpy
2 import ___________________  # for pyplot
```

Answer:

- `import numpy as np`,
- `import matplotlib.pyplot as plt`.

Problem III: Point out the issues (*Marks:* 30)

You need to write down *in your answer sheet* the locations (in the two programs) of the errors (line number), and how to correct each issue.
Marks: One for spotting each **syntax error** (there are 7 + 5), five for each **semantic error** (there are 2 + 2).

Q.III.1) This first program is applying the concepts of OOP (as seen in class and labs) to a small example of a banking software (two classes `account` and `savings` represent bank accounts). Hint: there are 5 typing mistakes (1 mark each) and 2 semantic mistakes (5 marks each). (**15 marks**)

Answer:

- Line 4, **syntax** error (#1/5): `self` is not passed as an argument,
- Line 10, **semantic** error (#1/2): for deposit, money should be added and not removed,
- Line 15, **syntax** error (#2/5): `int_rate` cannot be directly accessed, it has to be `account.int_rate`
- Line 18, **semantic** error (#2/2): `self.balance` will not be returned, `return interest, self.balance` should be used instead.
- Line 23, **syntax** error (#3/5): `self.n` is not defined, n is only given as an argument of the function,
- Line 29, **syntax** error (#4/5): `init` parameter is not given,
- Line 30, **syntax** error (#5/5): `account` class has no method `.change_accnt_num`, only the child class `savings` has.

```
1  class account():
2      int_rate = 0.09
3
4      def __init__(account_number, init_deposite):  # 1/5 typing
5          self.accnt_num     = account_number
6          self.init_deposite = init_deposite
7
8      def deposit(self, amount):
9          """ Method to add some money to an account."""
10         self.balance -= amount  # 1/2 semantic
11         return self.balance
12
13     def calc_interest(self):
14         """ Calculate half yearly interest, returns both interest ↩
               and latest balance."""
15         interest = self.balance * int_rate * 0.5  # 2/5 typing
16         self.balance += interest
17         return interest
18         return self.balance  # 2/2 semantic
19
20 class savings(account):
21     def calc_interest(self, n):
22         interest = self.balance * self.n * account.int_rate  # 3/5 ↩
               typing
23         return interest
24
25     def change_accnt_num(self, new_accnt_num):
26         self.accnt_num = new_accnt_num
27
28 # One example
29 my_account = account(1234)  # 4/5 typing
30 my_account.change_accnt_num(3456)  # 5/5 typing
```

Q.III.2) This first program is plotting the successive Taylor expansions of the exp function. **(15 marks)**

- For a function f, of class $\mathcal{C}^n$ at a point x_0 (for $n \geqslant 0$), we write $T_n(f, x_0)(x)$ the Taylor expansion for f at the point x_0 and order n ($T_n(f, x_0)(x)$ is a function of x).

  Mathematically, we recall that $T_n(f, x_0)(x)$ is defined as $\sum_{k=0}^{n} f^{(k)}(x_0)\frac{(x-x_0)^k}{k!}$.

- The program below is focusing on the function exponential ($f = \exp : x \mapsto \exp(x)$) and the point $x_0 = 0$. On your paper, write the expression of the first 4 Taylor expansions $T_0(\exp, x_0) = T_0$, $T_1(\exp, x_0) = T_1$, $T_2(\exp, x_0) = T_2$, $T_3(\exp, x_0) = T_3$.
- *Note* that because $\exp(x)$ is increasing quickly, we chose to restrict the domain for x in $[-3, 2]$.
- Locate and correct the semantic and typing mistakes.
  Hint: there are 5 typing mistakes (1 mark each) and 2 semantic mistakes (5 marks each).

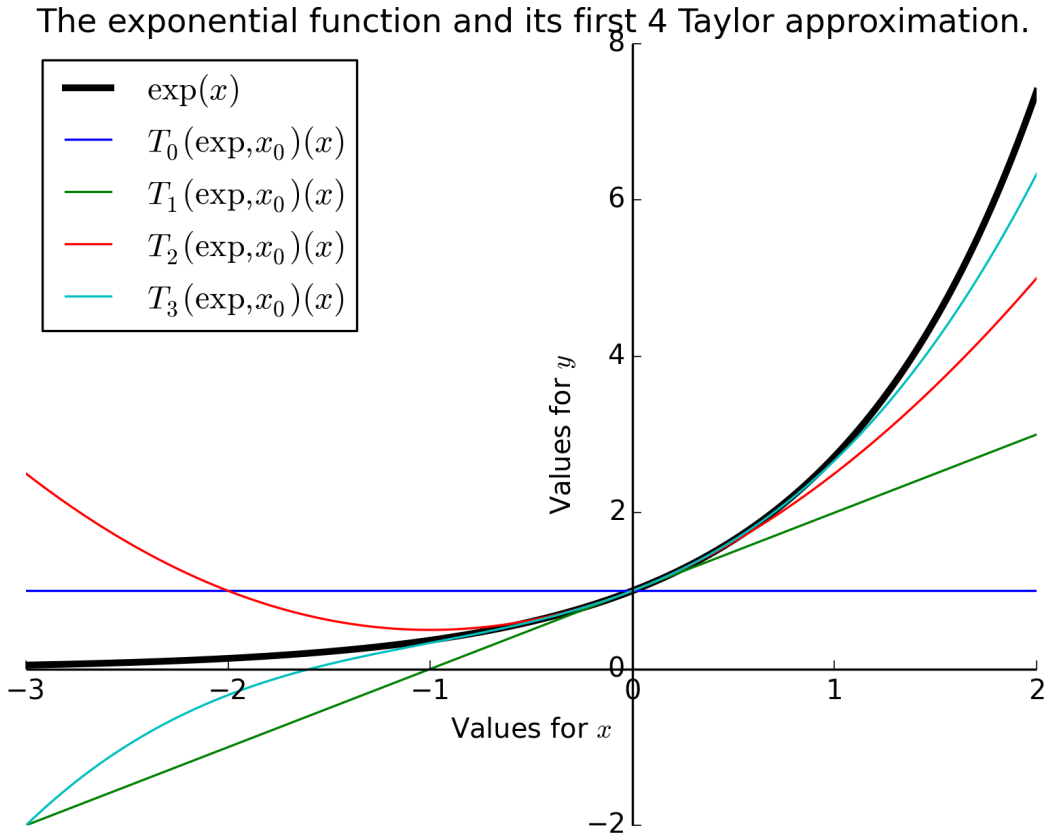
Answer:

- Line 11, **semantic** mistake (#1/2): `(x - x0) ** k` and not `(x0 - x) ** k` !
- Line 14, **typing** mistake (#1/5): we should return `y`, not `x`,
- Line 18, **semantic** mistake (#2/2): `-3, 2` and not `3, 2`,
- Line 21, **typing** mistake (#2/5): `plt.figure()`, not `plt.open_new_figure()`,
- Line 24, **typing** mistake (#3/5): `black` and not `baack`,
- Line 28, the given value to `plt.title` could also be interpreted as a mistake, as it is different from the one in the included picture! I think only one student remarked this. Sorry, but it was only a typing mistake.
- Line 35, **typing** mistake (#4/5): `ylabel` should be used, not `xlabel`,
- Line 40, **typing** mistake (#5/5): `dpi=180` should be used, not `resolution=180`.

```
1  # The numpy and pyplot packages have been imported (as usual)
2  from math import factorial
3
4  # The point and function we are interested about
5  x0 = 0.0
6  def f(x):
7      return np.exp(x)
8
9  def taylor_exp(x0, n, x):
10     y = (x - x0)**0
11     for k in xrange(1, n+1):
12         y += f(x0) * (x0 - x)**k / factorial(k)  # 1/2 semantic
13     return x  # 1/5 typing
14
15 # Samples for the X axis
16 X = np.linspace(3, 2, 500)  # 2/2 semantic
17 # New figure
18 plt.open_new_figure()  # 2/5 typing
19 # Plot exp(x)
20 plt.plot(X, f(X), color="baack", linewidth=3, label="$\exp(x)$")  # ↩
       3/5 typing
21
22 # Plot the successive Taylor expansion (can go up to 5 or more!)
23 for n in xrange(0, 4):
24     plt.plot(X, taylor_exp(x0, n, X),
25         label=("$T_" + str(n) + "(\exp, x_0)(x)$"))
26
27 # Title, xlabel and ylabel
28 plt.title("$\exp(x)$ and its first 4 Taylor approximations.")
29 # Here the title could also be interpreted as a mistake, as it is ↩
       different from the one in the included picture!
30
31 plt.xlabel("Values for $x$")
```

```
32 plt.xlabel("Values for $y$")  # 4/5 typing
33
34 # Add a legend (using the label of each plot), and show to graph
35 plt.legend(loc="upper left")
36
37 plt.savefig("Taylor_approx_of_exp.png", resolution=180)  # 5/5 ↩
       typing
38 # End of the program for plotting partial Taylor series for exp(x)
```

Remark: On Moondle, I uploaded `.gif` and `.mp4` animated views of these first Taylor approximations. Here is included a view of what the graphic looks like (if the program is correctly modified[1]):



Remark: For some of these mistakes, it can be really tricky to distinguish between a typic or a semantic mistake.

Therefore, we tried to be as nice and as soft as possible when we graded that problem.

[1] The x and y axis have been moved and centered to increase readability, but we do not ask you to do this.

Problem IV: computing the Lagrange polynomials (***Marks:*** **20**)

- TODO: add some extra explanations?

This last problem is focusing on an algorithm that can be used to numerically compute the Lagrange polynomials for an interpolation problem. .

- You need to think carefully when designing the algorithms, and then write them as *valid* Python programs,
- You will have to compute the time and memory complexities of the two functions, and justify your answers,
- The grading will not focus too much on syntax errors, but do your best to respect the Python syntax and write valid Python functions.
- Efficiency, conciseness and clarity of the code you write are also important.

Let f be a function of the real variable: $f : \mathbb{R} \to \mathbb{R}, x \mapsto f(x)$. By induction, for all integers $k \geqslant 1$, we define the k^{th} **divided differences** f_k of f, *recursively* like this:

1. (base case $k = 1$) For $x \in \mathbb{R}$, the first divided difference is $f_1[x] \overset{\text{def}}{=} f(x)$.
   (Please note the use of a bracket `[, ]` in this notation, to differentiate from the notation of a function).
2. (base case $k = 2$) For two distinct numbers $x, y \in \mathbb{R}$, the second divided difference is $f_2[x, y] \overset{\text{def}}{=} \dfrac{f(x) - f(y)}{x - y}$.
3. (induction case) Now, for $k \geqslant 1$, if f_k is defined, we can define the $(k+1)$-th divided difference by induction: If we have $(k + 1)$ distinct points $x_0, \dots, x_k \in \mathbb{R}$, we define $f_{k+1}[x_0, \dots, x_k]$ as:

$$f_{k+1}[x_0, \dots, x_k] = \frac{f_k[x_0, \dots, x_{k-1}] - f_k[x_1, \dots, x_k]}{x_0 - x_k}.$$

Let $n \geqslant 1$ and $x_0, \dots, x_n \in \mathbb{R}$ be $n + 1$ distinct real values. Let $L_{f,n}$ be the (unique and real) polynomial[2] of degree $\leqslant n$ which interpolates the function f on each of these points x_i: $\forall 0 \leqslant i \leqslant n, \quad L_{f,n}(x_i) = f(x_i)$.

The goal of this exercise is to *numerically* compute the value of this polynomial $L_{f,n}$ at a (arbitrary) point t (ie. the value $L_{f,n}(t)$).

Luckily, the divided difference operators ($X \mapsto f_k[X]$, as defined above), satisfy this identity:

$$\begin{aligned} \forall t \in \mathbb{R}, L_{f,n}(t) &= f_1[x_0] + f_2[x_0, x_1] \times (t - x_0) + f_3[x_0, x_1, x_2] \times (t - x_0)(t - x_1) \\ &\quad + \dots + f_{n+1}[x_0, \dots, x_n] \times (t - x_0) \dots (t - x_n) \\ \forall t \in \mathbb{R}, L_{f,n}(t) &= \sum_{k=1}^{n} \left(f_k[x_0, \dots, x_k] \times \prod_{i=0}^{k} (t - x_i) \right). \end{aligned} \tag{1}$$

Marking scheme: 8 marks for the function `divided_differences`, 4 marks for Q.IV.1.a) to Q.IV.1.d) (one each), and 8 marks for the second function `interpolate`.

Remark about the marking scheme: This problem was harder than the rest of the paper, and a small number of students tried it, and an even smaller number of you successfully wrote one of the two functions.

There, **we decided to be "very nice" about that problem**: if you tried something, you got some points. If we saw that you understood the basic concept (a recursive function, like the merge sort), you got more points.

Q.IV.1) Write a Python *function*, called `divided_differences`, that has to accept exactly 3 arguments: `X`, `Y` and `n`: (**8 marks**)

[2] Mathematically, we can prove that this polynomial, called the Lagrange polynomial, always exists (if the points x_i are distinct) and is unique, for **any** function f. But, no need for proving anything here.

- `X` will be a list of points $x_0, \dots, x_n$ (of size $n+1$);
- `Y` will be a list of values[3] taken by the function f: $y_0 = f(x_0), \dots, y_n = f(x_n)$.
- `n` is not really necessary, but it is convenient to have it as an argument, so we do not have to compute it too many times (indeed, `n = len(X) - 1 = len(Y) - 1`).

This function `divided_differences` has to **return** a list `D`, also of size $n+1$, containing the values of the successive divided differences for f (the ones that are used in the equation 1):

$$D[0] = f_1[x_0], \quad D[1] = f_2[x_0, x_1], \quad \dots, \quad D[n] = f_{n+1}(x_0, \dots, x_n).$$

Hint: You can chose for a naive recursive implementation of the function `divided_differences`, based on the inductive definition; or try to use a more efficient approach[4]. Be sure to include the base case correctly ($n = 0$, $n = 1$ – a typing mistake was left[5] in the exam paper, it said $n = 1$, $n = 2$ instead).

Solution for the first function: The task was mainly two things :

(a) handle the base cases ($n = 0$ or 1, ie a list of one point `X = [x0]` or two points `X = [x0, x1]`),

(b) and handle the recursive case.

```python
def divided_differences(X, Y, n):
    """ Returns a list D of size n+1, containing the values of the successive divided differences for the x points X = [x0,..,xn] and y points Y = [y0,..yn].

    - Time complexity is O(2^n).
    - Memory complexity is O(2^n).
    - This function is recursive, but we could also use dynamic programming (cf. Neuville's algorithm).
    """
    # Check that the n is correct
    assert n+1 == len(X) == len(Y)  # not required in the exam!
    if n == 0:
        # Base case k = 1
        # f_1[x0] = f(x0) = Y[0]
        return [Y[0]]
    elif n == 1:
        # Base case k = 2
        # f_2[x0, x1] = (f(x0) - f(x1)) / (x0 - x1) = (Y[0] - Y[1]) / (X[0] - X[1])
        return [Y[0], (Y[0] - Y[1]) / (X[0] - X[1])]
    else:
        # We write a naive recursive function

        # Left divided difference: f_{n-1}[x_0, .., x_{n-1}], ..., f_{n-1}[x_0, .., x_{n-1}]
        D1 = divided_differences(X[0:n-1], Y[0:n-1], n-1)

        # Right divided difference: f_{0}[x_1], ..., f_{n-1}[x_1, .., x_n]
        D2 = divided_differences(X[1:n], Y[1:n], n-1)

        # Now we can compute the new divided difference by using its definition
```

[3] Right now, your function shall take `Y` as a list of values, not compute the values by calling a function f.
[4] Similarly to what have been explained in class, e.g. for computing the terms of the Fibonacci's sequence.
[5] Sorry about that. You know, it can be really hard to eliminate all the typing mistakes.

```
28          fn = (D1[n-1] - D2[n-1]) / (X[0] - X[n-1])  # X[-1] is the same↩
                as X[n-1]
29
30          # We concatene the left difference list with the new one
31          return D1 + [ fn ]
```

Q.IV.1.a) What is the time complexity of your procedure `divided_differences` (as a function of n, order of the size of the input lists `X` and `Y`)? You may choose to make an illustration of the computational sequence to help you in calculating time complexity. (**1 mark**)

Q.IV.1.b) Can we hope to be more efficient than a time complexity of $O(n)$? (**1 mark**)

Q.IV.1.c) What is the memory complexity of your procedure `divided_differences` (as a function of n)? Hint: Try to count how many lists does your program use, and their sizes. (**1 mark**)

Q.IV.1.d) We clearly need at least one list of size $n+1$ (the list `D`, that is returned), but do we need any extra memory during the computation of its values? (**1 mark**)

Answer:

A.IV.1.a) The given solution is the "naive" recursive solution, and it is in $O(2^n)$). A good iterative solution, based on dynamic programming, can have a complexity of $O(n^2)$. More details can be found here: `http://www2.math.ou.edu/~npetrov/neville.pdf` or `https://en.wikipedia.org/wiki/Neville's_algorithm`

A.IV.1.b) Impossible, as we need to fill a list of size $n+1$ with some values that has to be computed iteratively (parallelism or vectorialization is impossible here). We have to fill the upper-triangle matrix for the divided differences, which take at least $O(n^2)$ operations. (An illustration can be found on the Wikipedia for Neville's algorithm.)

A.IV.1.c) Similarly, the recursive method has a memory complexity of $O(2^n)$, while the clever iterative method (dynamic program, cf. Neville's algorithm) will use only about $O(n^2)$ memory (half of a n by n matrix).

A.IV.1.d) If implemented correctly, we need a upper-triangle matrix for storing all the values for the divided differences, so at least a memory of $\frac{n(n+1)}{2} = O(n^2)$ is required.

Q.IV.2) Now write a function called `interpolate` that accepts exactly 3 arguments `X`, `f` and `t`: (**6 marks**)

- `X` will be a list of points $x_0, \dots, x_n$ (of size $n+1$). Note: now you need to define `n = len(X) - 1`.
- `f` is a function (like `math.exp` or `math.cos`). Note: now you need to compute the list of values `Y` by calling a function `f`: `Y[i] = f(X[i])` ($\forall 0 \leqslant i \leqslant n$).
- `t` is a real number (a `float` in Python).

Hint: this function `interpolate` should:

- First compute `n` and `Y`, by the method you like,
- Then use the previous function `divided_differences` to compute the list of coefficients `D`,
- And finally use the formula 1 (given above) to compute $L_{f,n}(t)$ thanks to these coefficients `D[0]`, `D[1]`, ..., `D[n]`, the values `x_0 = X[0]`, `x_1 = X[1]`, ..., `x_n = X[n]`, and the value of the point `t`.

Again, try to be as efficient as possible, but keep in mind that your program should be valid (ie. semantically correct: you compute what should be computed) and readable.

Similarly, quickly justify what are the time and memory complexity of this procedure `interpolate` (as a function of n the number of points). (**2 marks**)

Solution for the second function: In order to use the formula 1, we need a way to compute a product. It can be done with a `for` or `while` loop "manually", or by defining an additionnal function (called `prod` here).

```python
def prod(iterator):
    """ Compute the product of the values in the iterator.

    - Empty product is 1 (by convention).
    - And yes, weirdly, Python does not come with a built-in function prod.
    """
    current_product = 1
    for value in iterator:
        current_product *= value
    return current_product


def interpolate(X, f, t):
    """ Compute the value of the interpolation polynomial of order $n+1$ for $f$ on the points X = [x0, .., xn], ie the value $L_{f, n}(t)$.

    - Time and memory complexity is O(2^n) (because we use divided_differences).
    """
    # n is obvious to compute
    n = len(X) - 1
    # Y is just the list of values f(xi)
    Y = [ f(xi) for xi in X ]
    # We simply call the previous function
    D = divided_differences(X, Y, n)
    # And finally, we use the maths formula (1), directly:
    return sum(
        D[i] * prod(t - X[j] for j in xrange(0, i))
        for i in xrange(0, n+1)
    )
    # This formula uses two list comprehension, exactly as (1) used two Sigma or Pi symbols
```

Remark: This is *only for your curiosity*, **there is no question here**.

What we just did is writing a function that can be used to *predict* (or *estimate*) the value of a function f at a new (arbitrary) point t if the only knowledge that we have about f is its values on certain points $x_0, \dots, x_n$.

While this could seem to be useless, it is in fact important for lots of scientific applications: imagine that $y_i = f(x_i)$ is a set of points that you measured numerically, and you want to plot a smooth (polynomial) graph of the (unknown) function f as accurate as possible, well plotting the polynomial L_f is your best choice.

End of the exam paper.