CS 101: Introduction to Computer Science

Mahindra École Centrale

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

May 6th, 2015

Problem I: 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.

Q.I.1) We want to plot a graph of the two functions $f_1 : x \mapsto \tan(x) + 4$ and $f_2 : x \mapsto x^{10} - 5$, on the domain $[-1, 1]$ (with 400 points). f_1 has to be a continuous **blue** line, and f_2 has to be a **dotted red** line.

```
X = np.linspace(_________)
Y1 = np.tan(X) + 4
plt.plot(_________)  # For plotting (X, Y1)
Y2 = X**10 - 5
plt.plot(_________)  # For plotting (X, Y2)
```

Q.I.2) 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.

Q.I.3) 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.

Q.I.4) What is the *usual and recommended way* to import the `numpy` and `pyplot` packages for doing scientific computations and plotting?

```
import ___________________  # for numpy
import ___________________  # for pyplot
```

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

Q.I.6)

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

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

Q.I.7) 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.

Q.I.8) 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.

Q.I.9)

```
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?

Q.I.10)

```
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 _____.

Q.I.11)

```
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` = _____ ?

Q.I.12) 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 _____.

Problem II: MCQ (*Marks:* 30)

This first problem is a list of **Multiple Choice Question**. Each question (from Q.II.1 to Q.II.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 II: 1) a 2) c 3) d . . . 15) d* etc.

Q.II.1) 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`.

Q.II.2) 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.

Q.II.3) 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']`,

Q.II.4) 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.

Q.II.5) 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.

Q.II.6) 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.

Q.II.7) Which Python statement is used to force a specific exception?

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

Q.II.8) 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')`

Q.II.9) Asking Python to compute `a = 5 / '0'` will result in the exception

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

Q.II.10) Which of the following is not a specific characteristic of **Object Oriented Programming** (OOP)?

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

Q.II.11) 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]`.

Q.II.12) 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.

Q.II.13) 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`.

Q.II.14) What is the value of this Python expression `3*1**3`?

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

Q.II.15) 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.

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)**

```
1  class account():
2      int_rate = 0.09
3
4      def __init__(account_number, init_deposite):
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
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
16          self.balance += interest
17          return interest
18          return self.balance
19
20  class savings(account):
21      def calc_interest(self, n):
22          interest = self.balance * self.n * account.int_rate
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)
30  my_account.change_accnt_num(3456)
```

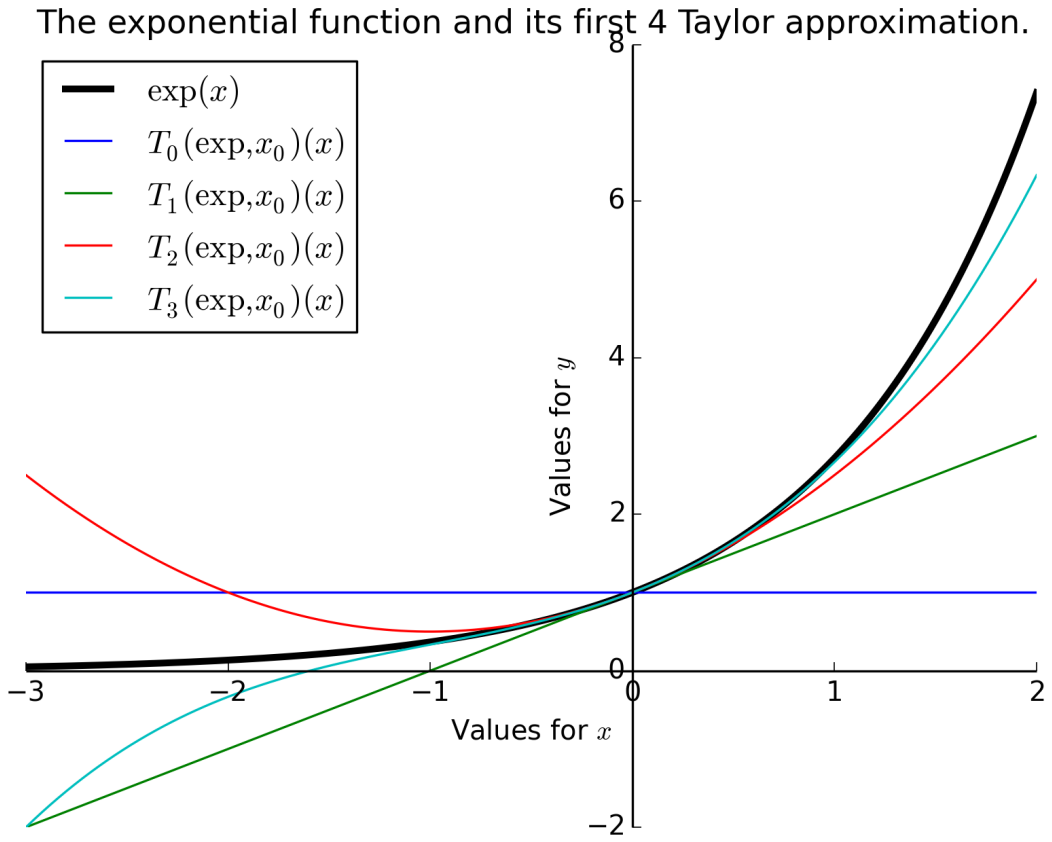
Q.III.2) This first program is plotting the successive Taylor expansions of the exp function. **(15 marks)**

- 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).

```
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)
13      return x
14
15  # Samples for the X axis
16  X = np.linspace(3, 2, 500)
17  # New figure
18  plt.open_new_figure()
19  # Plot exp(x)
20  plt.plot(X, f(X), color="baack", linewidth=3, label="$\exp(x)$")
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 plt.xlabel("Values for $x$")
30 plt.xlabel("Values for $y$")
31
32 # Add a legend (using the label of each plot), and show to graph
33 plt.legend(loc="upper left")
34
35 plt.savefig("Taylor_approx_of_exp.png", resolution=180)
36 # End of the program for plotting partial Taylor series for exp(x)
```

Here is included a view of what the graphic looks like (if the program is correctly modified[1]):



[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**)

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 $x_0, \ldots, x_k \in \mathbb{R}$, we define $f_{k+1}[x_0, \ldots, x_k]$ as:

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

Let $n \geqslant 1$ and $x_0, \ldots, x_n \in \mathbb{R}$ 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:

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

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`.

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

- `X` will be a list of points $x_0, \ldots, x_n$ (of size $n + 1$);
- `Y` will be a list of values[3] taken by the function f: $y_0 = f(x_0), \ldots, y_n = f(x_n)$.
- `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):

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

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 ($n = 1$, $n = 2$).

[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.

[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.

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**)

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**)

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.