Your submission for this assignment ***must include your full name*** *(as it appears on cuLearn) and you nine-digit* ***student number*** *at the top of* ***every file you submit****.* ***Source code submissions that crash*** *(i.e., terminate with an error) on execution will* ***receive a mark of 0****.*

Your submission for question 1 must be a ***single source file*** *with a* ***file name of either 'comp3007_w19_#########_a1_1.py' or 'comp3007_w19_#########_a1_1.java'*** *(with the number signs replaced by your nine-digit student number). It* ***must also be written*** *using either* ***Python 3*** *or* ***Java****. Python 2 will not be accepted, nor will any other language.*

Your submission for question 2 must be a ***single pdf file*** *with a* ***file name of 'comp3007_w19_#########_a1_2.pdf'*** *(with the number signs replaced by your nine-digit student number). It* ***must be written*** *using* ***Microsoft Word****,* ***Google Docs****, or* ***LaTeX****.* ***Photographs or scans*** *of handwritten submissions* ***will not be accepted*** *and will receive 0.*

Compress your submissions *into a* ***zip*** *file named* ***'comp3007_w19_#########_a1.zip'***

Late assignments will not be accepted *and will* ***receive a mark of 0****.*

The due date for this assignment is Saturday, January 26, 2019, by 11:00pm.

Question 1: "snake2pig" (10.0 marks)

For the first question of this assignment, you will design and write a program – in either Python 3 or Java (your choice) – that uses several recursive algorithms to produce a sequence of words in "Pig Latin" from an initial string in "Snake Case". You must **carefully** read the instructions below before proceeding.

A string that is written in "Snake Case" uses only lowercase letters and each word is separated by an underscore. Any word can be written in "Pig Latin" by removing all the characters in the word (starting from the left and moving to the right, up to but not including the leftmost vowel (specifically one of "A", "E", "I", "O", or "U") contained in the word. Without changing their order, these characters must be appended to the end of the word, and then the characters "AY" must be appended to the end of that.

By way of clarification, the word "ROBERT" would be transformed to "Pig Latin" by moving the "R" (i.e., all consonants from the left up to but not including the leftmost vowel) to the end of the word (to create "OBERTR") and then appending "AY" to produce "OBERTRAY". A word like THIRTY, in which there is more than one non-vowel character at the beginning of the word, would have the substring TH moved to the end, followed by "AY" to produce "IRTYTHAY" as the result. A word like EIGHT, in which the leftmost vowel is at position 0, would have no characters that must be moved to the end and would thus simply be "EIGHTAY".

The conversion of a string in "Snake Case" into "Pig Latin" requires transforming the initial "Snake Case" string into a list of words, and then transforming each word into "Pig Latin".

As a clarifying example, the "Snake Case" string:

```
"robert_is_thirty_eight"
```

would be parsed into a list of words (at the underscores):

```
["robert", "is", "thirty", "eight"]
```

and then each word would be converted to "Pig Latin":

```
["obertray", "isay", "irtythay", "eightay"]
```

The objective of this question is to have you revisit the paradigm shift we discussed in the first lecture, where you as the designer focus your efforts, not upon describing "how" a result might be computed but instead upon "what" each of the components is. Since we will (very shortly) be using recursive design exclusively, each of these definitions - except (e) - MUST be recursive in nature.

You are not permitted to use looping control structures anywhere in this program.

Your solution must meet the following requirements:

a) you must write a **recursive** function that will validate (i.e., return True or False) that the string passed as an argument does not use any uppercase letters,

b) you must write a **recursive** function that will take a string as an argument and produce an integer return value for the index of the leftmost underscore in the string,

c) you must write a **recursive** function that will take a string as an argument and produce a list of strings as a return value such that the argument string has been "split" at each instance of the underscore, using the function you wrote for (b) above,

d) you must write a **recursive** function that will take a string as an argument and produce an integer return value for the index of the leftmost vowel in the string,

e) you must write a function (which will **NOT** be recursive) that will take a string as an argument and produce a string return value by changing that "word" into "Pig Latin", using the function you wrote for (d) above, and

f) you must write a recursive **function** that will take a list of strings as an argument and produce a list of strings as a return value such that every element of the argument list has been passed to the function you wrote for (e) above.

Your interface need not be sophisticated - you can simply ask the user to type in the initial "Snake Case" string. Please include instructions for the teaching assistants (i.e., how they can test your program) as a README file or in the comments at the top of your submission.

The program described on the previous page (along with any other program submitted in this class) must be a completely original work, authored by you and you alone, prepared for this offering (i.e., Winter 2019) of COMP3007. Do not discuss this (or any other) question with anyone except the instructor or the teaching assistants, and do not copy materials from the internet or any other source. The exercises you complete for the second question of this assignment must also be completed individually.

Please also note that **you will only receive credit for code you have written** and you must **avoid using functions that you didn't write yourself** whenever possible. As a clarifying example, even though the "split" function could be used to separate a string into substrings, if you were to use the "split" function you would not receive the marks for item c) above, since you did not the write the code that accomplished the task. Other than the `charAt` and `substring` methods (if you are using Java), functions like `pop` or `remove` (for removing single elements from lists), and functions for input and output, you should not require any functions for this assignment that you did not write yourself.

Question 2: "Beta Reduction" (6.0 marks)

For this question, you must perform a β-reduction on each of the following expressions. Each of these expressions is VERY SIMILAR to some of the in-class examples that can be found in the lecture notes. The exercises you complete for the second question of this assignment must also be completed individually.

a) $\lambda a.\left(\lambda b.\left(\lambda c.\left(b\ ((a\ b)\ c)\right)\right)\right)\ \left(\lambda d.\left(\lambda e.d\ (d\ (d\ e))\right)\right)$

Hint: You must start by replacing all instances of a *that occur in* $\left(\lambda b.\left(\lambda c.\left(b\ ((a\ b)\ c)\right)\right)\right)$ *with the expression* $\left(\lambda d.\left(\lambda e.d\ (d\ (d\ e))\right)\right)$

b) $\lambda a.\left(\lambda b.\left(\lambda f.\left(\lambda x.\left((a\ f)((b\ f)x)\right)\right)\right)(\lambda f.(\lambda x.f\ x))\right)(\lambda f.(\lambda x.x))$

Hint: You must start by replacing all instances of a *that occur in* $\left(\lambda b.\left(\lambda f.\left(\lambda x.\left((a\ f)((b\ f)x)\right)\right)\right)(\lambda f.(\lambda x.f\ x))\right)$ *with the expression* $(\lambda f.(\lambda x.x))$

c) $\left(\left(\left(\lambda a.\left(\lambda b.\left(\lambda c.(\lambda d.(b\ c\ d)\ a)\right)\right)\ (\lambda f.(\lambda x.x))\right)(\lambda f.(\lambda x.f\ x))\right)(\lambda x.(\lambda y.y))\right)$

Hint: You must start by replacing all instances of a *that occur in* $\left(\lambda b.\left(\lambda c.(\lambda d.(b\ c\ d)\ a)\right)\right)$ *with the expression* $(\lambda f.(\lambda x.x))$