Foundations of Programming 2022-2023

Term1 Assignment – contains 100 marks (**25% of module**)

– PART A (40 marks) –

Consider the set containing *all* the strings that can be built by *repeating* the first five characters ($C_1\,C_2\,C_3\,C_4\,C_5$) of your Goldsmiths username an *arbitrary number of times n* (with $n>0$). For convenience, let us refer to this as to the set "*Names*".

For example, if your Goldsmiths username was "mjack002", then *Names*(mjack) would consist of the infinite set of strings {"mjack", "mjackmjack", "mjackmjackmjack", …, "mjackmjack….mjack"}. (Note: If your username contains only 4 letters followed by digits, the last character, C_5, will be a digit; if your username contains only 3 letters and then digits, C_4 and C_5 will both be digits; etc.).

Write a Python (Ver. **3.7.x**) program that asks the user to enter a string *s* and that outputs **True** if *s* belongs to the set *Names* (where *Names* is built using **the first five characters of your Goldsmiths username**, as described above), **False** otherwise.

USER INTERFACE SPECIFICATIONS

Here is an **example** of what a single run should look like (assuming, e.g., username mgara002):

(*on-screen output*): **Please enter the string to be checked:**
(*the user types in* "mgaramgaramgaramgaramgaramgaramgara")
(*on-screen output*): **True**. Goodye.

After printing the outcome, your code should *not* ask the user to enter another string, but simply stop. **Any departure from the above requirements will lead to a *loss of up to 40 marks*.**

NOTE: the string entered by the user can be of *any length*. In case the user enters an *empty string* (""), the program should not *crash*, but just print "False", and stop.

IMPORTANT: please read the "WARNING" message at the bottom of this page.

– PART B (60 marks) –

Assume that the latest Python release contains a new built-in function called '*Repeat-characters*' (or '®' for short) which, when applied to a string *s*, returns *all the strings* that can be built from *s* by *repeating any of its characters an arbitrary number of times n* ($n>0$).

For example, applying '*Repeat-characters*' to the string "Nice" produces the infinite set of strings **Nice**® = {"Nice", "NNice", "NNNice", "NNNNice", …, "N…Nice", "Niice", "Niiice", …, "Ni…ice", "N…Ni…ice", "Nicce", "Niccce", …, "Nic…ce", "N...Nic...ce", "Niic…ce", "Niiic…ce", …, "Ni…ic…ce", "N...Ni…ic…ce", "Nicee", "Niceee",…, "Nice…e", …., "N...Ni…ic…ce...e"}.

We call ***Names***® the set made up of (1) all the strings in *Names* (defined as in PART A above) plus (2) *all strings* built by applying ® to *an element of Names*. (Note that *Names*® is infinite).

Write a Python (Ver. **3.7.x**) program that asks the user to enter a string *s* and that outputs **True** if *s* belongs to the set *Names*® (where *Names* is built using **the first five characters of your Goldsmiths username**, as described in PART A), **False** otherwise.

The user interface requirements are analogous to those specified for PART A above.
Any departure from such requirements will incur in a *loss of up to 60 marks*.

WARNING: you are NOT allowed to use *any Python modules or libraries*. The presence of an 'IMPORT' statement in your Part-A or -B code will carry a penalty of up to 80 marks.

– MARKING SCHEME –

Your code will be assessed solely on its ability to correctly output **True** for all 'legal' strings – i.e., belonging to set *Names* for PART A and set *Names*® for PART B, both sets built using your Goldsmiths username (see p. 1) – and **False** for all others. It will be run on a set of tests:

- If your code for PART A does not run (e.g., because it contains a syntax error), or it does not output anything, or it outputs the incorrect result in 50% or more of the cases, it will *score less than 40 marks on this part, no matter how long, complex, "cool", or well-documented it is*;
- The score for PART B will be calculated by running your code on a set of 30 tests, containing a mix of "legal" and "illegal" strings. For each legal string correctly accepted, and for each illegal string correctly rejected (i.e., such that your code prints "True" or "False" as appropriate, and stops), **2 points** will be awarded. If your code crashes or fails to terminate (even after printing the correct result), or outputs the incorrect result, it will score **0 points** on that test. The final mark for PART B will be the sum of all points your code scores across the 30 tests. (Again, code length, complexity, 'elegance' or documentation will not be assessed here).

– TESTING YOUR CODE (especially PART B) –

You are strongly recommended to check that your code works prior to submitting it.

Below are a few **examples** of legal & illegal strings (built using a few hypothetical sample usernames) for which a correct solution to PART B should output **True** or **False**, respectively:

- **TESTING EXAMPLES for username** gmass001 (i.e., $\mathbf{C_1C_2C_3C_4C_5}$ = "gmass"):

  Here, the set *Names*®(gmass) would include – amongst others – the following legal strings:

  "gmassss", "gggmmmmmaaass", "gmassgmmmmassssssgmass", "gggmaassggmaaassgmass", "gggmmassgmasssgggggmmmmmaaaaaaaaassgmassgmmmmassssssss", …

  but *not* the following (illegal) strings, for which your code should return False:

  " ", "%asd ", "gmass ", "g.maaass", "gggmassgmaaaasS", "gmasgmass", "gmaaaaassgass", …

- **TESTING EXAMPLES for username** amoha004 (➔$\mathbf{C_1C_2C_3C_4C_5}$ = "amoha"):

  Here, the set *Names*®(amoha) would include – amongst others – the following legal strings:

  "amoha", "aaamohhhaaa", "amohaammmmoohaaaamohhhha", "aaamooooohhhaamohaamoha", "amohaaaamohhhaaaammmmooohaaaammmmmoooohhhhaamohhhhhaaaaaaa", …

  but *not* the following (illegal) strings, for which your code should return False:

  " ", "aa-moha ", "aMoha ", " amoha", "amohamoha", "ammmmohaaam", …

- **TESTING EXAMPLES for username** ttu001 (➔$\mathbf{C_1C_2C_3C_4C_5}$ = "ttu00"):

  Here, the set *Names*®(ttu00) would include – amongst others – the following legal strings:

  "ttu00", "tttu0000", "tttuuuu00ttttttttttu000000", "ttuuuuu00ttttttuuuuuu000000000ttu00", "ttu00ttuuuu00ttttttuuuuuu00000ttu00ttttttuuuu00ttttttttttttuuuuu00000000000", …

  but *not* the following (illegal) strings, for which your code should return False:

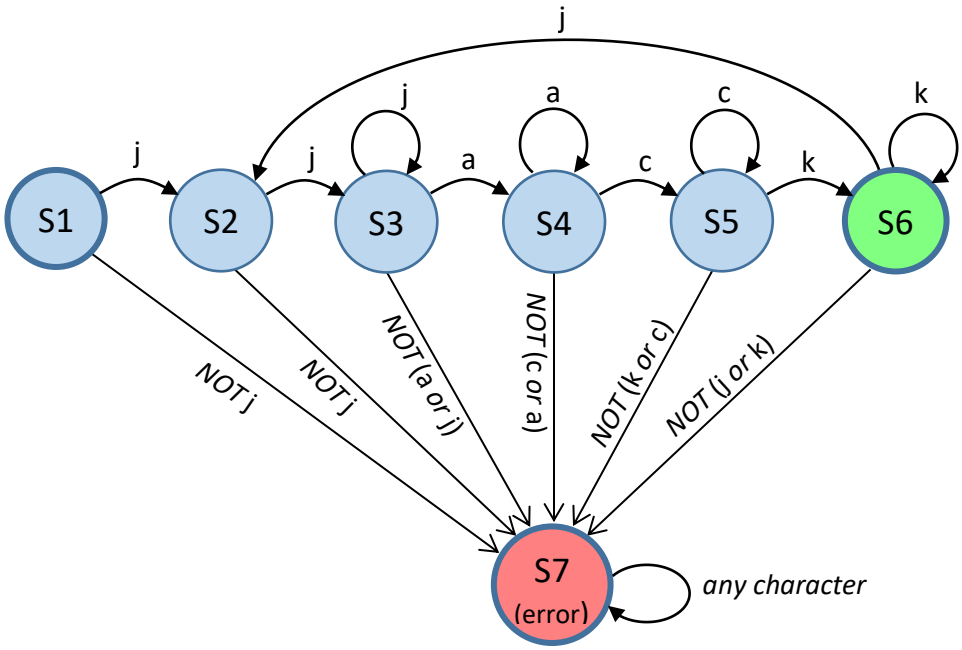
  "_", "ttu;00 ", " ttuuuu00", "ttu0", "ttu0o", "tu00", "ttuu00ttu000ttuu001", "tTuu000", …

NOTE: the above are (non-exhaustive) lists of examples built for a few 'made up' usernames; **you should write your own testing examples using your real Goldsmiths username (see p.1).**

WARNING: you are NOT allowed to use *any Python modules or libraries*. The presence of an 'IMPORT' statement in your Part-A or -B code will carry a penalty of up to 80 marks.

– OPTIONAL HELP / HINT FOR PART B –

Similar to a flowchart, a Finite State Acceptor (FSA) is a graph-like representation describing an algorithm that checks a given input string and decides whether it should be "accepted" or "rejected" (returning **True** or **False**, respectively). For example, consider the following FSA:

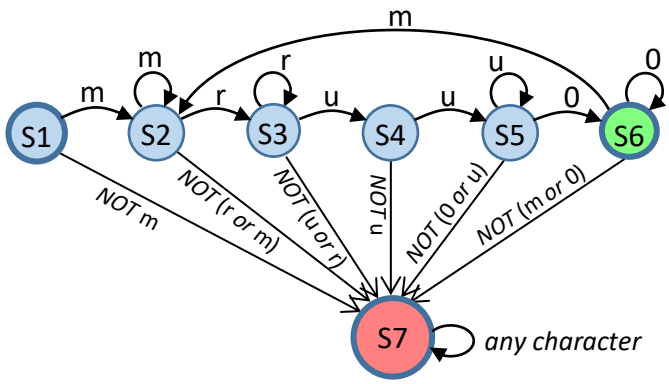


At any point in time, the system is in one of the states identified by the filled circles (here, S1, S2,…, S7). Each state is said to be either "*accepting*" (green colour fill) or "*non accepting*" (all other colours). The FSA always starts from **S1**. When given a string, the FSA scans its characters one by one, *from left to right*, and transitions from a state to the next depending on which character it reads (see letters on the 'arrows'). *After* the last character of the string has been read, the FSA stops: if it is in an 'accepting' state, the string is accepted, otherwise, rejected.

It can be seen that the above FSA accepts all and only the strings in the set Names®(jjack), i.e., as built from the first five characters of username "jjack002" ($C_1C_2C_3C_4C_5$ = "jjack").

HINT: To solve PART B, draw the FSA which accepts all and only the strings in the set *Names*® as built from your own Goldsmiths username (see p.1). *Your code can then simply 'mimic' the behaviour of the FSA: e.g., use a variable 'STATE' to store the current FSA's state...*

For example, here is the FSA for accepting the set *Names*® as built for username mruu002 (here, the five characters **$C_1C_2C_3C_4C_5$** are the first 4 letters, "mruu", plus the digit '0'):



WARNING: you are NOT allowed to use *any Python modules or libraries*. The presence of an 'IMPORT' statement in your Part-A or -B code will carry a penalty of up to 80 marks.