**Assignment Important Notes:**

**1) Explicit Regex Rules required to be delivered:**

| **Rule** | **Example** |
| --- | --- |
| Alteration | A | B |
| Concatenation | AB |
| 1 or more | A+ |
| 0 or more | A\* |
| Optional | A? |
| ORing (aka Char/Num classes) | [ABC] or [345] |
| Ranges | [0-9] or [a-z] |
| Brackets or grouping | (ABD)+ |

**2) Programming Language: Python**Actually any programming language that supports notebook scheme (code is written in cells), which is for simplicity, Python… you can use Kotlin if you are brave enough.

Why a notebook?  
= So the TA will go to the LAST cell and inject his input string.

**3) Code Editor: Colab**

Note: you can use any editor, but please make sure it is ready to be delivered on Colab.

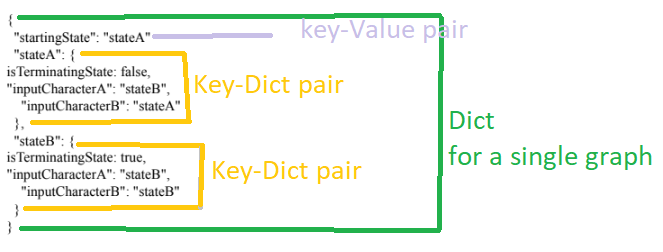
**4) Syntax for Last Cell:**

| RegexNFA\_Func(“<input regex>”) #this is just a template  NaynoonaX(“ Nay[a-zA-Z][0-9]+”)  #Here is a live example, Wrap all your logic in a **SINGLE** function by #any name, takes only 1 parameter, put in the very last cell ALONE. |
| --- |

**5) Outputs: 4 files**

Json & graph for 1st requirement + json & graph for 2nd requirement

**6) Syntax for Json Files is Strict as in the document, an automated script will run on them NOTICE:** each graph is a **dict**, the first element is always a key-value pair **(not a dict but a normal element)** like this: “starting state: A” the other elements are **dicts** and for each state dict, insert their info like mentioned (yes, including the starting state).



**7) A Recommendation for the Graphical package: Graphviz**

مجرد اقتراح للgraphical package, اسمها graphviz  
باكدج لطيفة لرسم الجرافز.

[اللينك](https://graphviz.readthedocs.io/en/stable/manual.html) دة فيه دوكمنتيشن ليها  
و طبعا احنا مبنقراش دوكمنتيشن فانت ابحث عالنت على اي توتوريال و امشي وراه الدنيا بسيطة مش صعبة....  
دة [توتوريال](https://analyticsindiamag.com/hands-on-guide-to-graphviz-python-tool-to-define-and-visualize-graphs/#:~:text=Graphviz%20is%20an%20open%2Dsource,the%20graph%20in%20DOT%20language) بسيط برضو

و لو عندك اقتراح تاني اكتبه في كومنت للناس كلها.

| **In NFA’s Json, if an edge goes to multiple destinations, put them in a list:**  **Example: ‘1’ : [‘S0’, ‘S1’, ‘S3’]**  **To dump a json file:**  **nfaOutFile = open('NFA.json', 'w', encoding="utf-8")**  **JsonObject = json.dump(outputJson, nfaOutFile,indent=6, ensure\_ascii=False)**  Full TestCases (nfa,dfa,minDFA, Jsons) in this folder:[CL assignment TestCases - Google Drive](https://drive.google.com/drive/u/0/folders/1q7Kbh7HW__C6K6Us4i4QVI2Z97vvfWsv)  Some further notes in this vid: [Compilers' Assignment | Part 3 : Further NOTES and Corrections](https://youtu.be/XCovfj1q_o0)  DFA minimization [[my humble exp](https://youtu.be/DFTiVY1nCqk)]  Same vid but from the foreign dude [[his exp](https://youtu.be/7W2lSrt8r-0)]  **Donia’s tips:**  **1- for checking the syntax of the Regex, we are allowed to use compile("regex") function from the python library "re" instead of implementing the validation from scratch.**  **2- the TA will evaluate the final output, i.e. the minimized DFA, so it doesn't matter if we added more epsilons in the NFA, or if we didn't all output the same NFA.**  **3- about the ranges: the TA told me that I can handle it as in the example here** [**https://i.ibb.co/gDys8vh/nfa2.png**](https://i.ibb.co/gDys8vh/nfa2.png)  **(i.e. there's no need to convert it to Oring)**  [**How to handle ranges video**](https://youtu.be/4y45TdKVfUQ)  [Humble explanation 😬](https://www.youtube.com/playlist?list=PLT7hbSdLHIDXpkQZo8meEKPad3jJQoi0m)  Kamoula’s Awesome [Plan](https://drive.google.com/drive/folders/18bI5AmWGWCZm398y3Cctpd0rYSJfY02y?usp=share_link) 😍  Intuition [video](https://www.youtube.com/watch?v=jMxuL4Xzi_A) on NFA → DFA, and [this](https://www.youtube.com/watch?v=7W2lSrt8r-0) for DFA minimization اجنبي  [aymanreda56/CL-Template (github.com)](https://github.com/aymanreda56/CL-Template) |
| --- |

Don't forget to remove spaces from the input regex

* **A very good** and simple approach to check the validity of the regex and to use it for NFA is called Infix-to-Postfix (aka shunting yard algorithm), which just reorders the string ensuring the inner most, very basic, highest precedence operations are on the leftmost side and hence computed first, no need for a tree, no need for recursive methods, no need for blackboxes.

Also, NFA makes very good use of the postfix format, very easy to get the NFA from postfix (lexers اصلا use it rather than trees).

1- [This](https://blog.cernera.me/converting-regular-expressions-to-postfix-notation-with-the-shunting-yard-algorithm/) simple tutorial explains the infix-to-postfix algorithm (which is part 1 of the requirement to check the validity of the regex),

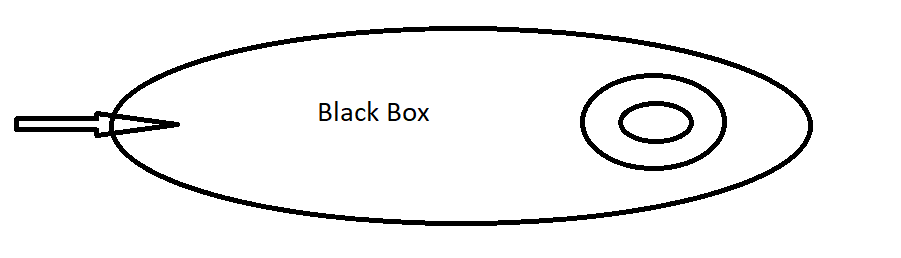
2- [This](https://medium.com/swlh/visualizing-thompsons-construction-algorithm-for-nfas-step-by-step-f92ef378581b) simple tutorial explains how to construct the NFA given the postfix format.

The repos below follow the same rules (infix-to-postfix-to-NFA-to-DFA)

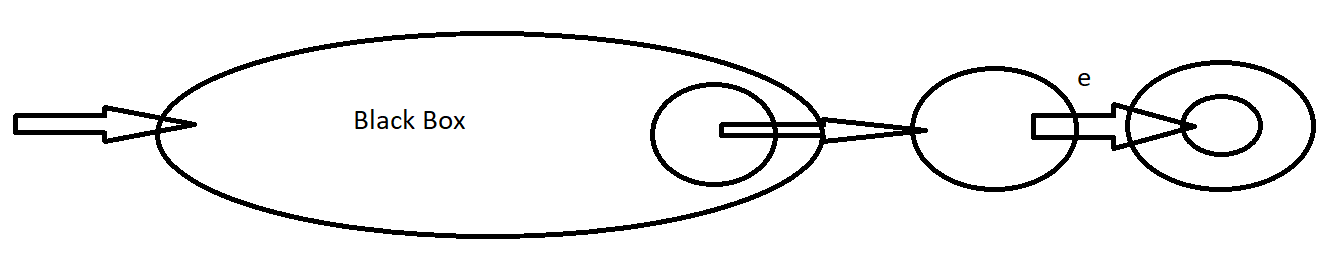
* [This](https://github.com/niemaattarian/Thompsons-Construction-on-NFAs) might help somebody, make sure to read the readme first
* For a non-coding video that helps in intuition, [this](https://www.youtube.com/watch?v=VbR1mGdP99s) seems a good beginner video (besides Essam’s Lecture).
* As I see, as a general rule you have to represent the input regex as a tree, you can use any representation of this tree, then take the very basic, very primitive (or deepest) elements in this tree and concat their corresponding ready-made NFA’s together (you will have already made some primitive NFA’s or as some of us call it, building blocks).
* For the checking the string validity part, you can either do some nested while and if checks, or you can implement a “infix-to-postfix” algorithm which is stack based, or a parse tree/regex 🙂
* Remember the TA asked to not perform any optimizations, this is for our favor, think about it.
* Ummm, I think [this](https://github.com/Megha-Bose/Automata-Theory-Conversions) repo is very similar to our assignment.

**Discussion:** When constructing the nfa, consider 2 approaches, either top down; taking a very big black box and beside it a very primitive element, then represent the element by its nfa block, then grabbing the black box and split it into another black box and a primitive element as so on…

Like this:



Then split this big black box into another black box and a primitive nfa:



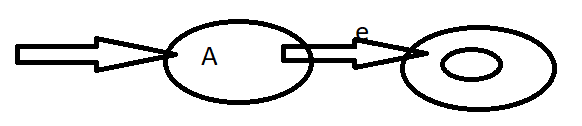
This approach is wiser but will have more epsilon edges.

Very friendly with the stack based approach that you used to parse the regex script.

You will face some ambiguities whether to split to the RHS or to the LHS.

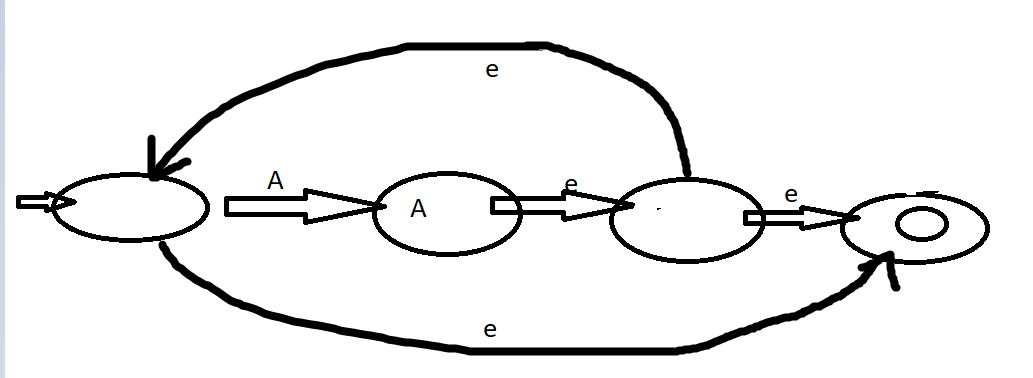
2nd approach bottom up:

Start with the innermost primitive nfa blocks then concat with higher and more general ones, like so:



Then concat this with another higher nfa block, CAUTION: will add some overhead to determine which node to connect with which node,

Consider the above example we want to wrap it inside a ()\* to become A\* ,  
So we must determine which node is the back edge and which node is the forward edge to become like this:



**~~To handle ranges ‘-’ and classes [abc], you can make a preprocessor step that puts | between characters, and replaces the ‘-’ with the characters in explicit format instead of the complex way I showed in the vid~~**

//if you had some implementation tips and tricks, plz write them in a comment.

Work hard, Play hard 😃

Have a chill in this park, with your cursor.

🌳🌳🌳🌳🌳🌳🌳🌳🌳🌳🌳🌳🌳🌳🌳🌳🌳🌳🌳🌳🌳🌳🌳🌳🌳🌳

🌳 🌳

🌳 🐝 🦝 🌲 🌳

🌳 🌳

🌳 🦋 🦩 🐕 🌳

🌳 🌳

🌳 🐐 🌲 🐧 🌳

🌳 🌳

🌳🌳🌳🌳🌳🌳🌳🌳🌳🌳🌳🌳🌳🌳🌳🌳🌳🌳🌳🌳🌳🌳🌳🌳🌳🌳