**How to use the code**

Go to the command line and type either: python Compiler.py or python Compiler.py file.txt. If you enter “python Compiler” you can manually enter anything to check if it is in a JSON format. If you use “python Compiler file.txt” (where “file” is any text file), it checks if the file is a properly formatted JSON file or not. Only one command-line argument should be passed to the script (in this case file.txt).

If it is a properly formatted JSON file, it will print out “Success: parsed input” in the console. If it is not a properly formatted JSON file, it will print out the “You have an error, Illegal character” and “Error: input is not properly formatted JSON” in the console.

I have created some test file called: jsonpass1.txt, jsonpass2.txt, jsonpass3.txt, jsonfail1.txt, jsonfail2.txt and jsonfail3.txt. I went to a JSON generator websites to obtain jsonpass.txt files and tested with my code and it successfully parsed it. For jsonfail.txt I wrote a text which is not in a JSON format and it failed to parse as it should.

I have also tested examples from json.org such as:

[] empty array or {} empty object 🡪 Passed

Invalid Unicode character u123G 🡪 Failed

Invalid key value pair … : … : … 🡪 Failed

[,,,,,,,,] or {,,,,,,,,} 🡪 Failed

**Detail about the code**

First I need to create my lexing rules. So I created a token list filled with tokens. By creating this list, we are telling lex that these token are valid. The tokens are number, comma, left and right bracket, left and right braces, strings, true, false and null. Next, I need to write my specification for each token, which we use a regular expression to define it. The regular expression for all token except strings and numbers are trivial. Strings are made up of zero or more, any Unicode character (except " or \) or any of this: \", \ \, \ /, \b, \ f, \n, \ r, \t or (\u plus 4 hexadecimal digits). The backslash(\) is known as an escape character. Most importantly, a string has a double quote (") at the start and at the end of it. So the regular expression for string token is:

*r'\"([^\\"]|(\\(("|\\|\/|b|f|n|r|t)|u[0-9A-Fa-f]{4})))\*"'*

Numbers can be an integer, decimal (float) or exponential (e^x). So the regular expression for number token is:

*r"-?(0|[1-9][0-9]\*)(\.[0-9]+)?([eE][+-]?[0-9]+)?"*

In my code t\_error is called if you get an error. The function prints out the illegal character and skips a token.

Finally, I write my parser rules.

* JSON is built on two data structure, which is array and object.
* An array is wrapped in brackets () and inside an array, there can be zero or more values which are separated by a comma.
* An object is wrapped in braces. Inside an object, there can be zero or more pair of key and value which are separated by a comma.
* A key in a key-value pair has to always be a string.
* The value in array or value in a key-value pair can be: String, number, array, object, true, false, null
* If I get an error while parsing, I raise Syntax error then "Error: input is not properly formatted JSON" is printed out and sys.exist(0) terminates the program.

For the random JSON generator I used this website:  
<https://onlinerandomtools.com/generate-random-json>