| **Number** | **Version** | **Title** | **Credits** | **Assessment** |
| --- | --- | --- | --- | --- |
| AS91906 | 1 | Use complex programming techniques to develop a computer program | 6 | Internal |

|  |
| --- |
| **Submission Checklist (Achieved)** |

To meet the achievement level criteria it is important that your code meet the minimum expectations of the assignment. Before submitting, please complete the checklist below to ensure you have not missed anything significant in your submission.

|  |  |
| --- | --- |
| Checklist | Done? Y/N |
| My program uses variables storing at least two types of data (e.g. numeric, text, Boolean) | Y |
| My program uses sequence, selection (IF) and iteration (LOOP) control structures | Y |
| My program uses Input from a user, sensors or another external source & produced output | Y |
| My program uses two or more complex programming techniques. | Y |
| My code is set out clearly using suitable whitespace | Y |
| I have included comments to document and explain what the code is doing | Y |
| I have prepared a video recording of my program working and will submit it with this document. | Y |
| I have completed the appropriate testing table(s) to show how effective my program is. | Y |
| I have copied my completed code into the space provided in this document | Never |
| I have prepared a video recording of my program working and will submit it with this document. | Y |

|  |
| --- |
| **About Your Project** |

In the space below, tell me anything about your program that I may need to know.

Include elements such as…

* What features did you add beyond the basic brief?
* Were there any changes to the brief that were agreed by your teacher?
* What environment is needed for your code to run?
* Etc.

In short, explain anything beyond hitting run button for a standard python environment and expecting the program to act as the example program provided.

|  |
| --- |
| Running example:  example\_regex\_viewer.exe should be provided (in /dist/). This executable is built with pyinstaller so should set up everything needed for the runtime environment  The example involves a quick, simple GUI that should highlight (in different colors) any phones and email addresses found by the regex engine (according to the example regular expressions I have provided; note that these regex are basic, just to show it working, and are not 100% reliable). The source code for this example can be found in /examples/regex\_viewer.py if you need.  Older details (possibly still relevant but requires more setup):  Need python >= 3.12 (match statements, @override) (tested on 3.12.3)  Program acts as a python package (module). It is not published on PyPI, so cannot be installed as easily as other packages. README.md contains further install instructions  To use debug graph, need  pip install networkx  pip install matplotlib  pip install pyqt5  pip install scipy  To use test suite’s HTML output, need  pip install pywin32  These are automatically installed by pip if the package is installed with the [debuggraphviewer] or [copy-html] options respectively. (See README.md, Building Manually section)  Program is a regular expression engine, constructing a deterministic finite automaton (a kind of state-machine) from a given regular expression. The DFA can efficiently (O(n) time) search strings.  A collection of examples are provided in `examples/`, which can be simply run once the required libraries are installed (note that only the regex\_viewer is bundled into pyinstaller, the other loibraries require manually setting up the environment)  To use as library, just `from regex import Regex` and then use the Regex constructor to compile a regex from a pattern, e.g `Regex(r"\w+(?:\.\w+)\*@\w+(\.\w+)+")` for a simple email regex. If this is stored (say, in a variable called rx), then it can be used as such: `rx.test(“my.email@gmail.com”)`, which should return True; or also `rx.match(“words [email@gmail.com](mailto:email@gmail.com) more words”)`, or `rx.replace\_in(“x [a@b.c](mailto:a@b.c) words”, “(email: %0)”)` (returning “x (email: [a@b.c](mailto:a@b.c)) words”). read the method docstrings for more info |

|  |
| --- |
| **1. Achieved Level Documentation** |

# 1.1 - 1.6

Evidenced within the code

# 1.7 Complex Tools

In the table below identify the complex tools you have used, why they have been used and where the evidence of their use can be found.

|  |  |  |
| --- | --- | --- |
| Complex Tool | Why was it used | Where is it used |
| *Student created classes/objects* | *To organise code and represent different constructs that make up the DFA structure (States (is a node) and Edges (are connections))* | *Everywhere* |
| GUI | To be able to actually see what is going on while debugging because otherwise debugging *frustrating* (i.e. I was crying while friends who saw it were like “ooo pretty picture”) bugs such as this:  (it was a simple regex it shouldn’t have exploded into that many states)  - would be entirely impossible as it would be impossible to keep track of all the states and edges in your head | debug\_graph\_viewer.py  rant:  turns out that theres a point when the code is complex enough that it does its own thing instead of what u were picturing  doesn’t help that its not deterministic because sets are unordered, and iteration order is thus random  the same result should always be achieved in the end, but the way it gets there is different each time so gl finding bugs :)  rant over |
| External libraries | To draw the gui because I wasn’t going to draw my own guis manually and do stuff such as picking a good layout to position the states in to make the graph more comprehendible; I didn’t have enough time to fix bugs AND make GUIs | debug\_graph\_viewer.py  uses networkx, matplotlib, scipy, pyqt5 |

|  |
| --- |
| **2.Merit Level Documentation** |

# 2.1 – Variable Names & Comments

Evidenced within code

# 2.2 - Conventions Used

Each programming language has a set of conventions that should be followed. For Python the current conventions are documented in the [PEP 8](https://www.python.org/dev/peps/pep-0008/) style guide and contain guidance such as:

* Variable and function names should all be lower case with words separated by underscores.
* Lines of code should not be longer than 79 characters
* Lines of comments should not be longer than 72 characters
* Functions should always contain a docstring

Other languages such as [C#](https://docs.godotengine.org/en/3.1/getting_started/scripting/c_sharp/c_sharp_style_guide.html) and [JavaScript](https://www.w3schools.com/js/js_conventions.asp) also have their own style guides.

In the table below identify the conventions you have followed and provide evidence that they have been applied.

|  |  |
| --- | --- |
| Selected Programming Language | Style Guide Used |
| Python | PEP(8 and others) |
| Evidence that Conventions have been applied | |
| *e.g.1 - A screen shot of the output from an online checking tool like* [**https://pep8plus.com/**](https://pep8plus.com/)*and / or* [**https://www.codewof.co.nz/style/**](https://www.codewof.co.nz/style/) **OR a not-online checker because im not crazy**  *e.g. 2 - A description of all of the conventions you have taken with examples. IM SORRY DID YOU SAY ALL????* | |
| Pylint Output:    All runs: [Pylint · Workflow runs · wntiv-main/regex (github.com)](https://github.com/wntiv-main/regex/actions/workflows/pylint.yml)  Note that many warnings have been disabled in the code (# pylint: disable=error-type), BUT ONLY where disabling those warnings is deemed to be valid. E.g. protected member use is often disabled for the internal library code, which should be able to access a value, but the end user shouldn’t (hence the member being protected). Some functions have many arguments, conditions, blocks, exit points (return), etc. Pylint flags “design” issues for these functions, but in cases where the function is deemed to be necessarily that large, these warnings have been disabled. This is common with very complex functions, often which cannot be refactored due to many nested loops which must pass a lot of state between them. (Trust me, I have considered this)  Conventions I have followed:  Indentation – 4 spaces (even tho tabs better - prove me wrong)  Example: my entire codebase  <- look spaces!!  Continuation lines either align to previous bracket like so:    (src/regex/regex\_factory.py, line 592)  OR are indented by one, relative to the parent line as such:    (src/regex/regex\_factory.py, line 597)  Line length – lines should be no more than 79 chars  e.g.    (src/regex/regex\_factory.py, line 592)    (src/regex/regex\_factory.py, line 597)  This makes code more readable, especially on narrow viewports (e.g. splitscreen, diff editor)  [Should a Line Break Before or After a Binary Operator?](https://peps.python.org/pep-0008/#should-a-line-break-before-or-after-a-binary-operator)  TLDR; before. E.g.    (src/regex/regex\_factory.py, line 541)  Blank lines:  Top-level class and functions surrounded with two blank lines  Methods with a signle blank line, etc.  e.g.    (src/regex/regexutil.py)  [Imports](https://peps.python.org/pep-0008/#imports)  e.g.    (tests/regex\_tests.py)  [Module Level Dunder Names](https://peps.python.org/pep-0008/#module-level-dunder-names)    (src/regex/\_\_init\_\_.py) (\_\_version\_\_ attr dynamically used for package deployment version)  [PEP 257 – Docstring Conventions | peps.python.org](https://peps.python.org/pep-0257/)  Everywhere  e.g.    Note: I have often skipped docstrings for \_\_init\_\_ when there are little/no arguments and/or the class docstring explains it well enough  No trailing whitespace  Example: well u cant see it, can you…?  In all seriousness, pylint warns about trailing whitespace, ergo, see above; no warning -> no trailing whitespace  Ok ive read enough PEP8 you get the gist  Oh yeah naming conventions:  PascalCase for classes  snake\_case elsewhere  begin with \_ for protected members  ALL\_CAPS for const  Example: | |

**NOTE: This section is intended to demonstrate that you are aware of and have adhered to the conventions for your selected language. A screen shot alone does not demonstrate that you have been successful throughout your entire project so make sure that you thoroughly check each aspect of your work.**

|  |
| --- |
| **3.Excellence Level Documentation** |

In the table below explain how you have addressed each of the refined characteristics of the assessment and where this can be seen in your program.

|  |  |  |
| --- | --- | --- |
| Refinements | Explanation | Where is it seen |
| 3.1 The program code is efficiently and effectively organised | API code is packaged into one big module, internally separated into separate files for organisation  Testing code is organised \*outside\* of the /src/ in a /tests/ folder, also organised into functionally separated submodules. It is outside of the /src/ folder so that it does not get packaged into a python package | /src/regex  /tests/ |
| 3.2 The program has features to help manage user input | Yeah if the user does smth wrong (only case is if they enter an invaid regex pattern) we raise an appropriate error (remember we are API so we want to alert API user with hard error if they e.g. input invalid regex) | Mostly in RegexBuilder.PatternParseError  In /src/regex/regex\_factory.py |
| 3.3 The program has user friendly output | 1 - its an API it doesn’t really have output, it returns a regex object which can be used in sensible ways and is well documented (= user-friendly?)  2 – there is always the debug graph viewer if the user is confused | Debug output in /src/regex/debug\_graph\_viewer.py  Api output in class Regex in /src/regex/regex.py |
| 3.4 The program includes error capture features | No it has error *raising* features in case the user does smth weird we raise an appropriate error with a detailed error message  (arguably it catches errors and re-raises them with a more detailed error message for the end user) | RegexBuilder.PatternParseError  In /src/regex/regex\_factory.py |
| 3.5 The program report unexpected behaviour to the user | I just told u twice already | RegexBuilder.PatternParseError  In src/regex/regex\_factory.py  Again |

|  |
| --- |
| **4.Final Submission & Testing** |

# 4.1 - Your Video

Record a video showing your game in operation with valid inputs. If you wish to show any error capturing you have included in your program, record these in a separate video.

Be sure to take your time and give the viewer of this video and opportunity to fully see your game in action. Save your video recording in a suitable file format (e.g. wmv or mp4 – NOT an iSpring file)

|  |
| --- |
| Below tell us the name of this file and where it is stored. |
| project\_root/docs/\*.mp4 (multiple videos if I forgot about things)   * example\_showcase is the main showcase, shows the features best, probably * build\_install\_and\_usage is an older video (not fully up-to-date but mostly still relevant) but shows off a lotg of the more in-depth install and build processes I have used while building this * debug\_graph\_viewer (also older video, but still mostly relevant) shows off how I have used the debug graph viewer during debugging. I used a hook inside of the regex library to show graphs at each optimisation step, but the debug graph viewer could also just be used to show only the final result, depending on what the user may need   if u cant find it, its in the same place on github, link is <https://github.com/wntiv-main/regex> |

# 4.2 – Demonstrate Authenticity

Evidenced by Teacher throughout project development & after submission

Git commit history also useful if u need ig

# 4.3 Valid “Expected” Input Testing (Achieved)

All unit test outputs over the development process can be seen at [Tests · Workflow runs · wntiv-main/regex (github.com)](https://github.com/wntiv-main/regex/actions/workflows/tests.yml)

Valid “Expected” tests are tests that show your program operates as expected if the inputs received are as you expect them to be. For each input test that your program works using valid inputs.

# EXPECTED

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test #** | **Test** | **Expected** | **Outcome** | **Result** |
| 0 | Test digits: Trying to construct regular expression from \A\d | Expected a DFA to be produced with the start state, followed by an \d-move to the end state. | Produced a DFA with the start state (1), followed by an \d-move to the end state (0). | PASS |
| 1 | Testing matches for \A\d\Z | Expected {'3', '0', '7', '9'} all to match, and {'five', 'hello', '?', 'a'} all to not match | As expected. | PASS |
| 2 | Test alphanum: Trying to construct regular expression from \A\w | Expected a DFA to be produced with the start state, followed by an \w-move to the end state. | Produced a DFA with the start state (1), followed by an \w-move to the end state (0). | PASS |
| 3 | Testing matches for \A\w\Z | Expected {'E', '7', '\_', 'a', '0', 'X', 'j'} all to match, and {'?', '+', '$', '-', '&'} all to not match | As expected. | PASS |
| 4 | Test string: Trying to construct regular expression from \Ahello | Expected a DFA to be produced with the start state, followed by an 'h'-move, followed by an 'e'-move, followed by an 'l'-move, followed by an 'l'-move, followed by an 'o'-move to the end state. | Produced a DFA with the start state (5), followed by an 'h'-move, followed by an 'e'-move, followed by an 'l'-move, followed by an 'l'-move, followed by an 'o'-move to the end state (4). | PASS |
| 5 | Test string reverse: Trying to construct regular expression from hello\Z, and reverse it | Expected a DFA to be produced with the start state, followed by an 'o'-move, followed by an 'l'-move, followed by an 'l'-move, followed by an 'e'-move, followed by an 'h'-move to the end state. | Produced a DFA with the start state (5), followed by an 'o'-move, followed by an 'l'-move, followed by an 'l'-move, followed by an 'e'-move, followed by an 'h'-move to the end state (0). | PASS |
| 6 | Testing matches for \Ahello\Z | Expected 'hello' to match, and {'dsnjkdf', 'he', 'helloijwlk'} all to not match | As expected. | PASS |
| 7 | Testing matches for \Ahello\Z | Expected 'olleh' to match, and {'', 'ollehjkdsfk', 'dsnjkdf', 'oll', 'hello'} all to not match | As expected. | PASS |
| 8 | Test star: Trying to construct regular expression from \Aa\* | Expected a DFA to be produced with the end state, followed by an 'a'-move to the end state. | Produced a DFA with the end state (0), followed by an 'a'-move to the end state. | PASS |
| 9 | Testing matches for \Aa\*\Z | Expected {'', 'a', 'aaa', 'aaaaaa'} all to match, and {'hello', 'njds'} all to not match | As expected. | PASS |
| 10 | Test epsilon closure and minification: Trying to construct regular expression from \A(a?b?c?)\* | Expected all epsilon-moves to be removed. As such the DFA should start with the end state, followed by an {'b', 'a', 'c'}-move to the end state. | Produced a DFA with the end state (0), followed by an {'b', 'a', 'c'}-move to the end state. | PASS |
| 11 | Test epsilon closure and minification with or: Trying to construct regular expression from \A(a|b|c)\* | Expected all epsilon-moves to be removed. As such the DFA should start with the end state, followed by an {'b', 'a', 'c'}-move to the end state. | Produced a DFA with the end state (0), followed by an {'b', 'a', 'c'}-move to the end state. | PASS |
| 12 | Test plus: Trying to construct regular expression from \Aa+ | Expected a DFA to be produced with the start state, followed by an 'a'-move to the end state, followed by an 'a'-move to the end state. | Produced a DFA with the start state (1), followed by an 'a'-move to the end state (0), followed by an 'a'-move to the end state. | PASS |
| 13 | Testing matches for \Aa+\Z | Expected {'aaaaaa', 'a', 'aaa'} all to match, and {'', 'aaaaaabdsjmn', 'hello', 'njds'} all to not match | As expected. | PASS |
| 14 | Test optional: Trying to construct regular expression from \Aa? | Expected a DFA to be produced with the start state, followed by:   * an 'a'-move to the end state. * an ε-move to the end state. | Produced a DFA with the start state (1), followed by:   * an 'a'-move to the end state (0). * an ε-move to the end state. | PASS |
| 15 | Testing matches for \Aa?\Z | Expected {'', 'a'} all to match, and {'aaa', 'aaaaaaa'} all to not match | As expected. | PASS |
| 16 | Test or: Trying to construct regular expression from \A(?:a|b) | Expected a DFA to be produced with the start state, followed by an {'b', 'a'}-move to the end state. | Produced a DFA with the start state (1), followed by an {'b', 'a'}-move to the end state (0). | PASS |
| 17 | Testing matches for \A(?:a|b) | Expected {'b', 'a'} all to match, and {'d', 'c'} all to not match | As expected. | PASS |
| 18 | Test longer or: Trying to construct regular expression from \Ahello|\Agoodbye | Expected a DFA to be produced with the start state, followed by:   * an 'h'-move to a state, followed by an 'e'-move, followed by an 'l'-move, followed by an 'l'-move, followed by an 'o'-move to the end state. * an 'g'-move to a state, followed by an 'o'-move, followed by an 'o'-move, followed by an 'd'-move, followed by an 'b'-move, followed by an 'y'-move, followed by an 'e'-move to the end state. | Produced a DFA with the start state (11), followed by:   * an 'h'-move to state (0), followed by an 'e'-move, followed by an 'l'-move, followed by an 'l'-move, followed by an 'o'-move to the end state (4). * an 'g'-move to state (5), followed by an 'o'-move, followed by an 'o'-move, followed by an 'd'-move, followed by an 'b'-move, followed by an 'y'-move, followed by an 'e'-move to the end state. | PASS |
| 19 | Test reverse or: Trying to construct regular expression from hello\Z|goodbye\Z, and reverse it | Expected a DFA to be produced with the start state, followed by:   * an 'o'-move to a state, followed by an 'l'-move, followed by an 'l'-move, followed by an 'e'-move, followed by an 'h'-move to the end state. * an 'e'-move to a state, followed by an 'y'-move, followed by an 'b'-move, followed by an 'd'-move, followed by an 'o'-move, followed by an 'o'-move, followed by an 'g'-move to the end state. | Produced a DFA with the start state (11), followed by:   * an 'o'-move to state (4), followed by an 'l'-move, followed by an 'l'-move, followed by an 'e'-move, followed by an 'h'-move to the end state (0). * an 'e'-move to state (10), followed by an 'y'-move, followed by an 'b'-move, followed by an 'd'-move, followed by an 'o'-move, followed by an 'o'-move, followed by an 'g'-move to the end state. | PASS |
| 20 | Test powerset construction: Trying to construct regular expression from \A(?:hello|hi) | Expected two h-moves to be merged into one. As such the DFA should start with the start state, followed by an 'h'-move to a state, followed by:   * an 'e'-move to a state, followed by an 'l'-move, followed by an 'l'-move, followed by an 'o'-move to the end state. * an 'i'-move to the end state. | Produced a DFA with the start state (5), followed by an 'h'-move to state (3), followed by:   * an 'e'-move to state (0), followed by an 'l'-move, followed by an 'l'-move, followed by an 'o'-move to the end state (4). * an 'i'-move to the end state. | PASS |
| 21 | Test complex powerset construction: Trying to construct regular expression from \A(?:\da|[13579a-e]b) | Expected the edges to be merged into an equivilentedge that covers both of them. As such the DFA should start with the start state, followed by:   * an {'8', '0', '6', '4', '2'}-move to a state, followed by an 'a'-move to the end state. * an {'3', '9', '7', '1', '5'}-move to a state, followed by an {'b', 'a'}-move to the end state. * an {'a', 'c', 'b', 'e', 'd'}-move to a state, followed by an 'b'-move to the end state. | Produced a DFA with the start state (4), followed by:   * an {'8', '0', '6', '4', '2'}-move to state (0), followed by an 'a'-move to the end state (2). * an {'3', '9', '7', '1', '5'}-move to state (3), followed by an {'b', 'a'}-move to the end state. * an {'a', 'c', 'b', 'e', 'd'}-move to state (1), followed by an 'b'-move to the end state. | PASS |
| 22 | Testing matches for \Ahello\Z|\Agoodbye\Z | Expected {'hello', 'goodbye'} all to match, and {'he', 'helloodbye', 'hellodbye'} all to not match | As expected. | PASS |
| 23 | Testing matches for \A(hello|goodbye)\Z | Expected {'eybdoog', 'olleh'} all to match, and {'', 'og', 'eybdo', 'hello', 'goodbye'} all to not match | As expected. | PASS |
| 24 | Test class specifier: Trying to construct regular expression from \A[ab] | Expected a DFA to be produced with the start state, followed by an {'b', 'a'}-move to the end state. | Produced a DFA with the start state (1), followed by an {'b', 'a'}-move to the end state (0). | PASS |
| 25 | Testing matches for [ab] | Expected {'b', 'a'} all to match, and {'d', 'c'} all to not match | As expected. | PASS |
| 26 | Test inverted class specifier: Trying to construct regular expression from \A[^ab] | Expected a DFA to be produced with the start state, followed by an -{'b', 'a'}-move to the end state. | Produced a DFA with the start state (1), followed by an -{'b', 'a'}-move to the end state (0). | PASS |
| 27 | Testing matches for [^ab] | Expected {'d', 'c'} all to match, and {'b', 'a'} all to not match | As expected. | PASS |
| 28 | Test class specifier range: Trying to construct regular expression from \A[c-g] | Expected a DFA to be produced with the start state, followed by an {'f', 'g', 'c', 'e', 'd'}-move to the end state. | Produced a DFA with the start state (1), followed by an {'f', 'g', 'c', 'e', 'd'}-move to the end state (0). | PASS |
| 29 | Testing matches for [c-g] | Expected {'f', 'e', 'd'} all to match, and {'k', 'z', 'a', 'r', 'b'} all to not match | As expected. | PASS |
| 30 | Test quantifier: Trying to construct regular expression from \Aa{3} | Expected a DFA to be produced with the start state, followed by an 'a'-move, followed by an 'a'-move, followed by an 'a'-move to the end state. | Produced a DFA with the start state (3), followed by an 'a'-move, followed by an 'a'-move, followed by an 'a'-move to the end state (2). | PASS |
| 31 | Testing matches for a{3} | Expected 'aaa' to match, and {'a', 'ccccc', 'bbb'} all to not match | As expected. | PASS |
| 32 | Test min quantifier: Trying to construct regular expression from \Aa{3,} | Expected a DFA to be produced with the start state, followed by an 'a'-move, followed by an 'a'-move, followed by an 'a'-move to the end state, followed by an 'a'-move to the end state. | Produced a DFA with the start state (3), followed by an 'a'-move, followed by an 'a'-move, followed by an 'a'-move to the end state (2), followed by an 'a'-move to the end state. | PASS |
| 33 | Testing matches for \Aa{3,} | Expected {'aaaaaaaa', 'aaaa', 'aaa'} all to match, and {'a', 'ccccc', 'bbb'} all to not match | As expected. | PASS |
| 34 | Test max quantifier: Trying to construct regular expression from \Aa{,3} | Expected a DFA to be produced with the start state, followed by:   * an ε-move to the end state. * an 'a'-move to a state, followed by:   + an ε-move to the end state.   + an 'a'-move to a state, followed by:     - an ε-move to the end state.     - an 'a'-move to the end state. | Produced a DFA with the start state (3), followed by:   * an ε-move to the end state (1). * an 'a'-move to state (0), followed by:   + an ε-move to the end state.   + an 'a'-move to state (2), followed by:     - an ε-move to the end state.     - an 'a'-move to the end state. | PASS |
| 35 | Testing matches for \Aa{,3}\Z | Expected {'aa', 'a'} all to match, and {'aaaaaa', 'bbb'} all to not match | As expected. | PASS |
| 36 | Test minmax quantifier: Trying to construct regular expression from \Aa{3,5} | Expected a DFA to be produced with the start state, followed by an 'a'-move, followed by an 'a'-move, followed by an 'a'-move to a state, followed by:   * an ε-move to the end state. * an 'a'-move to a state, followed by:   + an ε-move to the end state.   + an 'a'-move to the end state. | Produced a DFA with the start state (5), followed by an 'a'-move, followed by an 'a'-move, followed by an 'a'-move to state (2), followed by:   * an ε-move to the end state (3). * an 'a'-move to state (4), followed by:   + an ε-move to the end state.   + an 'a'-move to the end state. | PASS |
| 37 | Testing matches for \Aa{3,5}\Z | Expected {'aaaa', 'aaa'} all to match, and {'aaaaaa', 'aa', 'bbb'} all to not match | As expected. | PASS |
| 38 | Testing matches for \A(?P\w+(?:\.\w+)\*)@(?P\w+(?:\.\w+)+)\Z | Expected {'abc@cashmere.school.nz', 'a@b.c', 'abc12@gmail.com', 'my\_name@outlook.com', 'hynescj20@cashmere.school.nz'} all to match, and {'user@invaliddomain.', '', 'invalid.@gmail.com', '.invalid@gmail.com', 'user@.invaliddomain', 'not\_a@validdomain'} all to not match | As expected. | PASS |
| 39 | Testing matches for \A(?:\+\d{1,2}\s\*)?\(?\d{3}\)?[\s.\-]?\d{3}[\s.\-]?\d{4}\Z | Expected {'1234567890', '+64 022 345 6789', '123.456.7890', '123 456 7890', '(123) 456-7890', '+123456789999', '+91 (123) 456-7890', '123-456-7890'} all to match, and {'', '+()--', '41568739037463'} all to not match | As expected. | PASS |
| 40 | Testing matches for hello | Expected {'chello', 'hello, world', 'Bob says "hello"', 'reujdengvjkfdmkjhelloewoijkmlkf'} all to match, and {'shelf lo', 'shell', 'Hello', 'hjello', 'shell of', 'hell', 'olleh'} all to not match | As expected. | PASS |
| 41 | Testing matches for hello | Expected to find matches in  fred says hello, bob says hello  at 10:15, 26:31; and in  hello there, chello  at 0:5, 14:19 | As expected. | PASS |
| 42 | Testing matches for (?P\w+(?:\.\w+)\*)@(?P\w+(?:\.\w+)+) | Expected to find matches in  --- Bob --- Bob is an exceptional employee. He works passionately and very efficiently completes tasks before deadlines. You can contact Bob at bob.ross@gmail.com. Alternatively, phone him at +12 987 456 3212.  --- Fred --- Fred is a highly valued student at Example University. They achieve well above average in most their assignments. Fred has a strong work ethic and is helpful and friendly when working with team members. You can get in touch with him at fred1998@x.com, or +99 (121)232-4656.  at 144:162 (should match bob.ross@gmail.com), 460:474 (should match fred1998@x.com) | As expected. | PASS |
| 43 | Testing matches for (?:\+\d{1,2}\s\*)?\(?\d{3}\)?[\s.\-]?\d{3}[\s.\-]?\d{4} | Expected to find matches in  --- Bob --- Bob is an exceptional employee. He works passionately and very efficiently completes tasks before deadlines. You can contact Bob at bob.ross@gmail.com. Alternatively, phone him at +12 987 456 3212.  --- Fred --- Fred is a highly valued student at Example University. They achieve well above average in most their assignments. Fred has a strong work ethic and is helpful and friendly when working with team members. You can get in touch with him at fred1998@x.com, or +99 (121)232-4656.  at 192:208 (should match +12 987 456 3212), 479:496 (should match +99 (121)232-4656) | As expected. | PASS |

# 4.4 Boundary Input Testing (Merit)

Boundary tests are tests that show your program operates as expected if the inputs received are at the extreme ends of the allowed input range. It is also worth including the other side of each boundary to show that the boundaries are actually operating correctly. E.g. To test an input asking for a number between 1 and 10, apply 4 tests: 1 then 10, but also 0 and 11!

Works very well for *numerical* data, with *well-defined ordering*, however not so well for *categorical* data, such as the strings I am working with… so very limited boundaries to test.

# BOUNDARY

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test #** | **Test** | **Expected** | **Outcome** | **Result** |
| 0 | Testing matches for \Aa\*\Z | Expected '' to match | As expected. | PASS |
| 1 | Testing matches for \Aa+ | Expected 'a' to match, and '' to not match | As expected. | PASS |
| 2 | Testing matches for [c-g] | Expected {'g', 'c'} all to match, and {'b', 'h'} all to not match | As expected. | PASS |
| 3 | Testing matches for \Aa{3}\Z | Expected 'aaa' to match, and {'aa', 'aaaa'} all to not match | As expected. | PASS |
| 4 | Testing matches for \Aa{3,} | Expected {'aaaa', 'aaa'} all to match, and 'aa' to not match | As expected. | PASS |
| 5 | Testing matches for \Aa{,3}\Z | Expected {'aa', 'aaa'} all to match, and 'aaaa' to not match | As expected. | PASS |
| 6 | Testing matches for \Aa{3,5}\Z | Expected {'aaaaa', 'aaa'} all to match, and {'aaaaaa', 'aa'} all to not match | As expected. | PASS |
| 7 | Testing matches for hello | Expected {'hello world', 'hello', 'say hello'} all to match, and {'this is hell', 'ello there'} all to not match | As expected. | PASS |
| 8 | Testing matches for hello | Expected to find matches in  hello world  at 0:5 (should match hello); and in  say hello  at 4:9 (should match hello); and in  hello  at 0:5 (should match hello) | As expected. | PASS |

# 4.5 Comprehensive Invalid Input Testing (Excellence)

**Comprehensive testing needs to show testing of all areas of the program.**

Invalid tests are tests that ensure the program operates as expected regardless of the inputs made by the user. This will be the biggest of all of the testing sections.

No, it wont be the biggest, because there is very little invalid input testing you can do when the only input is the regular expression pattern string that the end user passes to the API

# INVALID

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test #** | **Test** | **Expected** | **Outcome** | **Result** |
| 0 | Building regular expression from invalid pattern \*abc | Expected PatternParseError to be raised. | Exception was raised:  '\*' must be preceded by another token at position 0: "\*abc"  ^- here | PASS |
| 1 | Building regular expression from invalid pattern +abc | Expected PatternParseError to be raised. | Exception was raised:  '+' must be preceded by another token at position 0: "+abc"  ^- here | PASS |
| 2 | Building regular expression from invalid pattern ?abc | Expected PatternParseError to be raised. | Exception was raised:  '?' must be preceded by another token at position 0: "?abc"  ^- here | PASS |
| 3 | Building regular expression from invalid pattern |abc | Expected PatternParseError to be raised. | Exception was raised:  '|' must be preceded by another token at position 0: "|abc"  ^- here | PASS |
| 4 | Building regular expression from invalid pattern abc| | Expected PatternParseError to be raised. | Exception was raised:  '|' must be succeeded by atleast one token at position 3: "abc|"  ^- here | PASS |
| 5 | Building regular expression from invalid pattern [ | Expected PatternParseError to be raised. | Exception was raised:  Could not find closing ']' for opening '[' at position 0: "["  ^- here Caused by ValueError: substring not found | PASS |
| 6 | Building regular expression from invalid pattern [abc | Expected PatternParseError to be raised. | Exception was raised:  Could not find closing ']' for opening '[' at position 0: "[abc"  ^- here Caused by ValueError: substring not found | PASS |
| 7 | Building regular expression from invalid pattern hello] | Expected PatternParseError to be raised. | Exception was raised:  Unopened ']' at position 5: "hello]"  ^- here | PASS |
| 8 | Building regular expression from invalid pattern [a-] | Expected PatternParseError to be raised. | Exception was raised:  range with no end char at position 2: "[a-]"  ^- here | PASS |
| 9 | Building regular expression from invalid pattern [-a] | Expected PatternParseError to be raised. | Exception was raised:  range with no start char at position 1: "[-a]"  ^- here | PASS |
| 10 | Building regular expression from valid pattern [a\-] | Expected PatternParseError not to be raised | As expected. | PASS |
| 11 | Building regular expression from valid pattern [\-a] | Expected PatternParseError not to be raised | As expected. | PASS |
| 12 | Building regular expression from invalid pattern a{hello} | Expected PatternParseError to be raised. | Exception was raised:  Invalid n-quantifier numeral at position 2: "a{hello}"  ^- here | PASS |
| 13 | Building regular expression from invalid pattern a{1,2,3} | Expected PatternParseError to be raised. | Exception was raised:  Invalid n-quantifier: should have no more than two values at position 5: "a{1,2,3}"  ^- here | PASS |
| 14 | Building regular expression from invalid pattern a{0} | Expected PatternParseError to be raised. | Exception was raised:  Invalid n-quantifier: max must be > 0 at position 2: "a{0}"  ^- here | PASS |
| 15 | Building regular expression from valid pattern a{1} | Expected PatternParseError not to be raised | As expected. | PASS |
| 16 | Building regular expression from invalid pattern a{-1,} | Expected PatternParseError to be raised. | Exception was raised:  Invalid n-quantifier numeral at position 2: "a{-1,}"  ^- here | PASS |
| 17 | Building regular expression from valid pattern a{0,} | Expected PatternParseError not to be raised | As expected. | PASS |
| 18 | Building regular expression from invalid pattern a{,0} | Expected PatternParseError to be raised. | Exception was raised:  Invalid n-quantifier: max must be > 0 at position 3: "a{,0}"  ^- here | PASS |
| 19 | Building regular expression from invalid pattern a{5,3} | Expected PatternParseError to be raised. | Exception was raised:  Invalid n-quantifier: max is smaller than min at position 2: "a{5,3}"  ^- here | PASS |
| 20 | Building regular expression from valid pattern a{3,5} | Expected PatternParseError not to be raised | As expected. | PASS |

# Your Code

Please copy the final code from your program into the space below.

|  |
| --- |
| Final Code |
| Yeah right like that will help  <https://github.com/wntiv-main/regex> if you want |

## Reflection thing ig I just want to quickly talk a few more things

All in all project turned out pretty decent, it seems to work for all my test cases however I am still not entirely confident to release it for use in production code, because while it does work, it is kind of janky :/ While developing, I have kept performance in mind quite a bit, because while executing a DFA is generally super performant (the regex\_viewer example has almost no noticeable input latency, supporting this claim); Constructing and optimising the regex \*can\* be quite expensive as it requires iterating the entire graph structure. This can sometimes cause noticeable latency in constructing the regex (dependent on the complexity of it; seen in the time before the window loads in the example regex\_viewer), but keep in mind that once it is constructed, it should execute fairly performantly (O(n) time on the length of the string (ignoring the fact that I haven’t fully optimised execution (think of it like I am “emulating” the DFA, not fully compiling and executing it), in the optimal case I would be compiling the DFA to machine code, using jump tables or range comparisons (e.g. chr > ‘a’ && chr < ‘z’); whichever is most optimal. This would give a super performant DFA execution, however that is outside the scope of this project. The project is designed to be extensible to allow such expansions of it if need be)).

## Retrospective mistakes

* some ParserPredicates being mutable, meaning that there needed to be two different hash/equality comparisons, one for set memberism (since set’s don’t work well with mutable objects) and another for actual comparison etc. – This could have been replaced with immutable PargserPredicates which, when needing to be modified, could be removed from the set, mutated (or replaced with a mutated version, in monad style) and re-inserted into the set. This would only need to occur in ONE place, but now the cursed mutability is sprinkled all around the codebase
* The need for the ConsumeString ParserPredicate – what was wrong with just using ConsumeAny with a single item? Sure, its slightly less efficient, but the code for segregating the two is ugly